1-1-2005

A bi-directional analysis technique for software safety and software security

Qian Feng
Iowa State University

A bi-directional analysis technique for software safety and software security

by

Qian Feng

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Robyn R. Lutz, Major Professor
Carl K. Chang
Ratnesh Kumar

Iowa State University

Ames, Iowa

2005

Graduate College
Iowa State University

This is to certify that the master's thesis of

Qian Feng

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

The author would like to thank her advisor Dr. Robyn R. Lutz for her contribution to this thesis, her remarkable advice, wisdom, patience, understanding, and constant support. The author also thanks Josh Dehlinger for his comments and useful discussions. Thanks to all the lab mates in the Laboratory for Software Safety in Department of Computer Science at Iowa State University: Josh Dehlinger, Janet (Jing) Lu, Oko Swai, Kendra Schmid, Barbara Nsian, and HongYu Sun for the support, suggestions, friendship, and fun during the two years of the author's master program study. This research was supported in part by National Science Foundation grants 0204139 and 0205588.

CHAPTER 1 INTRODUCTION

With the recent rapid development of software technology, safety-critical and security-critical software is playing a more important role in people's lives. An example of a safety-critical and security-critical system is a smart door system. A smart door system recognizes the users' identifications, gives commands to the door to open or close, and also prevents intruders from entering [Cook, 2003; Youngblood, 2002; Smart-home project, Finland; Fellbaum and Hampicke]. Another example of such a system is the network protocols used to transfer messages between spacecrafts, base stations, and satellites in deep space. The messages transferred include uplinked commands to spacecraft and downlinked science data and images. Such networks experience scheduled intermittent connectivity, long signal delays, and frequent communication interruptions. Because of the importance of the data and the commands as well as the severe power and/or memory constraints in deep space, the safety and security of these networks need even more attention from software engineers. Unfortunately, the highly successful protocols and security strategies of today's Internet operate poorly in such environments [Cert et al., 2005; Scott, 2005; Symington, 2005; Burlegigh, 2005; Ramada, 2005; Fall, 2005; Warthman, 2003].

The importance of system safety and system security has promoted much research on systematic techniques to develop complete safety and security requirements [Bosch, 2000; Bass, 2003; Leveson, 1995; Dehlinger and Lutz, 2004; Dehlinger and Lutz, 2005; Feng and Lutz, 2005; Lu and Lutz, 2002; Liu and Lutz, 2005; Crook et al., 2005; Jürjens, 2005; Lancy and Nuseibeh, 2004; Lin et al., 2003; Mead, 2005; Moffett et al., 2004; Stavridou, 1998; Viega, 2005; Weber, 2005]. Safety requirements and security requirements share many similarities. For instance, both are non-functional requirements, dealing with threats or risks; both often involve negative requirements that may conflict with some important functional requirements; and, both involve requirements for which testing alone is insufficient to prove their correctness and completeness [Landwehr, 1984; Leveson,

1995; Leveson 1986; Stavridou, 1998; Srivatanakul, 2004]. Clearly, each also involves problems and techniques that apply specifically to it.

Several researchers have explored the relationship between safety analysis and security analysis and applied the methods used traditionally on each to the other. For example, HAZOP, which has been used in safety analysis, is also applied to security analysis [Foster, 2005; Srivatanakul et al., 2004]. Among the techniques used in the analysis of the software safety, bi-directional analysis has shown promise in security requirement analysis [Lutz and Woodhouse, 1997]. This method combines a forward search from potential failure modes to their effects with a backward search from feasible hazards to the contributing causes of each hazard. The forward search is similar to a Software Failure Modes and Effects Analysis (SFMEA); the backward search is similar to a Software Fault Tree Analysis (SFTA). The combination of the forward and backward search has proven effective in discovering latent safety requirements.

This thesis uses bi-directional analysis to investigate the requirements for two applications in the areas of safety analysis and security analysis. The two contributions of this work both involve the application of the bi-directional analysis and develop systematic methods to apply it to these two different types of non-functional requirements analysis. The first application, in Chapter 2, is to construct a systematic safety requirements analysis technique for a smart door product line. The final results include a reusable safety analysis and the discovery of missing safety requirements. The second application, in Chapter 3, investigates a systematic security requirements technique for a Delay Tolerant Network protocol called the Bundle Protocol. This work improves an existing security analysis technique by integrating it with the bi-directional analysis to demonstrate and challenge the correctness and completeness of the resulting security requirements specifications. This thesis also reports the discovery of missing security requirements and the remediation of the security requirements.

Both applications explore the technique of applying bi-directional analysis to software safety analysis and software security analysis and find that the bi-directional analysis assists in finding incorrect and incomplete requirements.

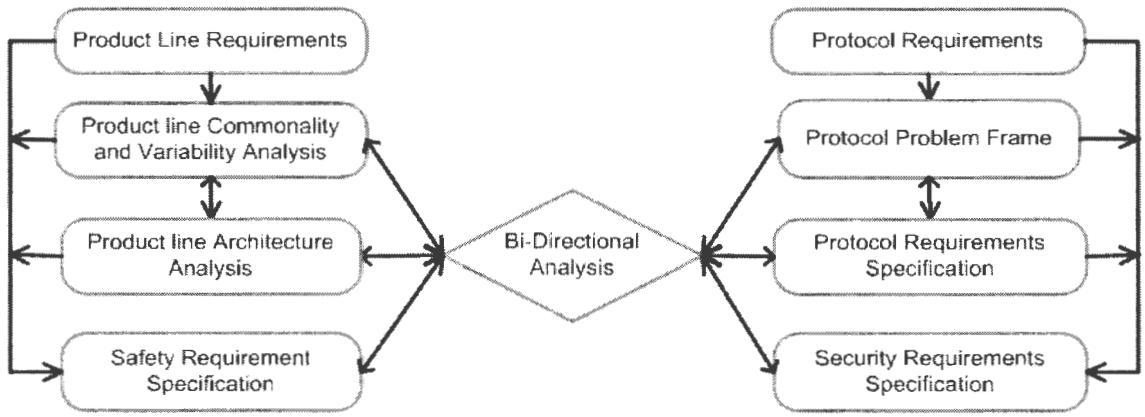

Figure 1 Overview of the Work Presented in the Thesis

Figure 1 gives an overview of these two portions of work and illustrates the commonalities and differences between them. The diamond represents the bi-directional analysis, and the rectangles represent other sub-process steps. The connectors show the relationships (the data flow) among sub-processes. The left side of Figure 1 is the safety requirements analysis technique developed in Chapter 2 of this paper; the right side of Figure 1 is the security analysis technique shown in Chapter 3 of this paper.

The difference between the applications lies primarily in the inputs to the bi-directional analysis. In the safety requirements analysis, the inputs to the bi-directional analysis are the results from the product-line commonality and variability analysis, the product-line architectural design, and the product-line safety requirements specifications. However in the security requirements analysis, the inputs to the bi-directional analysis are the Problem Frame, the requirements specification, and the security requirements specifications [Moffett et al., 2004].

Note that the connectors between the bi-directional analysis and other processes are double-arrowed, showing that not only the bi-directional analysis uses other sub-processes' results as input

but also that the results from the bi-directional analysis will be input to other processes and help develop the completeness and the correctness of other sub-processes. For example, bi-directional analysis analyzes the safety requirements specification, and the result of the bi-directional analysis will be the remediation of missing or incomplete safety requirements. The rest of this section introduces each of these two pieces.

The work presented in Chapter 2 applies a systematic safety requirements analysis to a safety-critical product line. A method is constructed to apply bi-directional analysis to the software product line. The results show that the bi-directional analysis found missing and incorrect software safety requirements and demonstrate that the proposed method can handle the challenges to safety analysis posed by variations within a product line.

A product line is a set of systems developed from a common set of core requirements and sharing a suite of common traits among the members [Ardis and Weiss, 1997; Weiss and Lai, 1999]. This research describes results from an investigation into how, and to what extent, product-line safety analyses can be performed and reused as a product-line asset. Reuse of software assets currently includes product-line requirements specifications, product-line core architecture, product-line test suites, and product-line performance analyses. This work investigated two major challenges to extending the bi-directional analysis method to product lines: how to adequately understand and specify the safety consequences of the variations among the members of the product line, and how to structure the process such that the safety analysis is derived from, and traceable to, the product-line requirements and design. To solve the challenges, the standard artifacts of the product-line domain-engineering process were used, including: (1) the Commonality and Variability Analysis that specifies both the requirements shared by all the systems in the product line and the variations among the systems' requirements; (2) the product-line architecture that forms the shared, core software architecture for all the systems and supports the required variations; and (3) the product-line use

cases and scenarios that specify the range of uses and the sequences of events that some or all of the systems in the product line may experience.

The work presented in Chapter 3 applies the software security requirements analysis to a security-critical network protocol called the Bundle Protocol used in a Delay Tolerant Network. Here, an extended bi-directional analysis method was developed to assess the correctness and the completeness of the security requirements. The results also show how the bi-directional analysis found missing and incorrect software security requirements

A Delay Tolerant Network (DTN) is an end-to-end network providing communications in challenging environments such as deep space communication or sensor-based networks. Existing network protocols are not sufficient to fulfill the requirements associated with the highly stressed environments. Research on DTN has produced a DTN architecture and the proposal, specification, and implementation for the Bundle Layer Protocol and the LickLider Protocol [Cert et al., 2005; Scott, 2005; Symington, 2005; Burlegigh, 2005; Ramada, 2005; Fall, 2005; Warthman, 2003]. Because of the importance of the role that security is playing in DTN [Ramada, 2005], a thorough analysis of the completeness and correctness of the protocol requirements should be done as early as possible.

The approach that this thesis uses for security requirements analysis is based on the Framework of Core Security Requirements Artefacts approach (FCSRA) proposed by Moffett, Haley, and Nuseibeh [Moffett et al., 2004]. It utilizes Jackson's Problem Frame to analyze system functional requirements and security goals and to lead to the security requirements specifications [Moffett et al., 2004]. The Problem Frame illustrates the machine (software system)'s behavior through specifying the interactions between the machine and other environment component (domains) around the machine and provides a means of analyzing and decomposing problems [Jackson, 2001; Haley et al., 2004].

Although the FCSRA approach has the advantages of being easy to apply and understand, traceable, and reusable, it has the drawback that it does not give a convincing demonstration of the correctness of the resulting security requirements specification. To address this point, the work reported here applied an extended bi-directional analysis and demonstrated some vulnerabilities that challenge the trust assumptions of the Problem Frame. Instead of using the standard artifacts of a domain engineering process, such as system architecture, use cases, and scenarios, the extended bi-directional analysis method takes the Problem Frame and the behavior descriptions of the machine as input. The results for security showed missing security requirements and proposed four more security requirements remediation to prevent damage from message flooding.

By applying the bi-directional analysis, the work in this paper provides structured methods and step-by-step guidelines for improving the safety and security of software. Evaluation of the technique showed that the bi-directional analysis can help find incompleteness in the safety requirements and the security requirements.

The reminder of the paper is organized as follows. Chapter 2 is joint work with R. Lutz and has been published in [*Journal of Systems and Software*, 2005, 78, 111-127]. It develops a systematic method to do safety requirements analysis on a product line. Chapter 3 develops a systematic method to do security requirements analysis on the Bundle Protocol.

CHAPTER 2 BI-DIRECTIONAL ANALYSIS OF PRODUCT LINE

A paper published in *The Journal of Systems and Software*[1]

Qian Feng[2,3] and Robyn R. Lutz[4]

Abstract

As product-line engineering becomes more widespread, more safety-critical software product lines are being built. This paper describes a structured method for performing safety analysis on a software product line, building on standard product-line assets: product-line requirements, architecture, and scenarios. The safety-analysis method is bi-directional in that it combines a forward analysis (from failure modes to effects) with a backward analysis (from hazards to contributing causes). Safety-analysis results are converted to XML files to allow automated consistency-checking between the forward and backward analysis results and to support reuse of the safety-analysis results throughout the product line. The paper demonstrates and evaluates the method on a safety-critical product line subsystem, the Door Control System. Results show that the bi-directional analysis method found both missing and incorrect software safety requirements. Some of the new safety requirements affected all the systems in the product line while others affected only some of the systems in the product line. The results demonstrate that the proposed method can handle the challenges to safety analysis posed by variations within a product line.

---

[1] Reprinted with permission of *J. of Systems and Software*, 2005, 78, 111-127

[2] Graduate student, Department of Computer Science

[3] Primary researcher and author

[4] Author for correspondence

1 Overview

As product-line engineering becomes more common, more safety-critical product lines are being built. A product line is a set of systems developed from a common set of core requirements and sharing a suite of common traits among the members [Ardis and Weiss, 1997; Weiss and Lai, 1999]. Examples of safety-critical product lines include embedded medical devices such as pacemakers, space telescopes, power-plant control systems, and some industrial robots.

The potential for reuse among the systems in a software product line extends beyond code reuse. Reuse of software assets currently includes product-line requirements specifications, product-line core architecture, product-line test suites and product-line performance analyses.

This paper describes results from an investigation into how, and to what extent, product-line safety analyses can be performed and reused as a product-line asset. That is, we are interested in the potential for reuse of the safety analysis among the members of a safety-critical product line. The motivation for this research is to improve the safety-analysis techniques available to developers of commercial, safety-critical product lines.

It is important to note that safety is a property of a single system, not of a set of systems. Thus, any safety analysis done during the early domain engineering of the product line (i.e., when the entire product line is being defined) must be re-evaluated, adjusted, and completed during application engineering (i.e., when each individual system is built). Some preliminary results regarding the reuse of safety analyses during application engineering have appeared in [Dehlinger and Lutz, 2004; Lu and Lutz, 2002]. In this paper we focus instead on the process of developing a product-line safety analysis for the domain engineering phase of safety-critical software product lines.

The paper extends the Bi-directional analysis (BDSA) method [Lutz and Woodhouse, 1997] to product lines. The BDSA method combines a forward search from potential failure modes to their effects with a backward search from feasible hazards to the contributing causes of each hazard. The

forward search is similar to a Software Failure Modes and Effects Analysis (SFMEA); the backward search is similar to a Software Fault Tree Analysis (SFTA). The combination of the forward and backward search has proven effective in discovering latent safety requirements.

The work described in this paper investigates two major challenges to extending the BDSA method to product lines: how to adequately understand and specify the safety consequences of the variations among the members of the product line, and how to structure the process such that the safety analysis is derived from, and traceable to, the product-line requirements and design.

In order to address these challenges in a way that is likely to be used by industry, the safety-analysis method presented in this paper is grounded in the standard artifacts of the product-line domain-engineering process. These domain-engineering assets are: (1) the Commonality and Variability Analysis that specifies both the requirements shared by all the systems in the product line and the variations among the systems' requirements; (2) the product-line architecture that forms the shared, core software architecture for all the systems and supports the required variations; and (3) the product-line use cases and scenarios that specify the range of uses and the sequences of events that some or all of the systems in the product line may experience.

Grounding the safety analysis in the domain-engineering products has several benefits. First, it supports documented traceability from the extended commonality analysis to the safety analysis and is requisite for future automated updating of the safety analysis as the product line evolves. Second, linking the safety analysis to the products that capture the subtleties of the domain provides more complete handling of variations, the rationales for the variations and the consequences of the variations in the safety analysis. Third, using standard domain-engineering assets promotes readier adoption of the safety-analysis method by companies building safety-critical, software product lines and can lower the cost of performing enhanced safety analyses on these product lines. The first two benefits are demonstrated in the paper by application of the safety-analysis method to the Door

Control System, a safety-critical subsystem of the Smart Home product line.

Figure 1 shows an overview of the analysis method developed in this paper with the Extended Commonality Analysis driving the bi-directional analysis in the lower half of the figure. Our method consists of seven steps:

Step 1: Performed Commonality and Variability Analysis to specify the requirements for the given product line.

Step 2: Developed the architectural design and sequence diagrams from the product-line requirements.

Step 3: Extended the Commonality and Variability Analysis based on the results of the Commonality and Variability Analysis and the Architecture Design diagrams.

Step 4: Constructed the Software Fault Tree Analysis from the results of the Extended Commonality and Variability Analysis.

Step 5: Constructed the Software Failure Mode and Effect Analysis from the results of the Extended Commonality and Variability Analysis.

Step 6: Translated the results of the Software Fault Tree Analysis and Software Failure Mode and Effect Analysis into XML files.

Step 7: Compared the resulting XML files using Xlinkit.

The rest of the paper is organized as follows: Section 2 presents related work. Section 3 describes the product line commonality and variability analysis. Each step in this and subsequent sections is illustrated with examples from the Door Control System. Section 4 shows how to extend the commonality analysis with information from the architectural design and the scenarios' sequence diagrams. Section 5 describes how the bi-directional software safety analysis uses the extended commonality analysis to guide and structure it. Section 6 evaluates the method, both by automated consistency checking of the forward vs. backward safety analysis results, and by discussing the

missing or incomplete software safety requirements for the product line found by this method. Section 7 briefly summarizes the results and provides some concluding remarks.

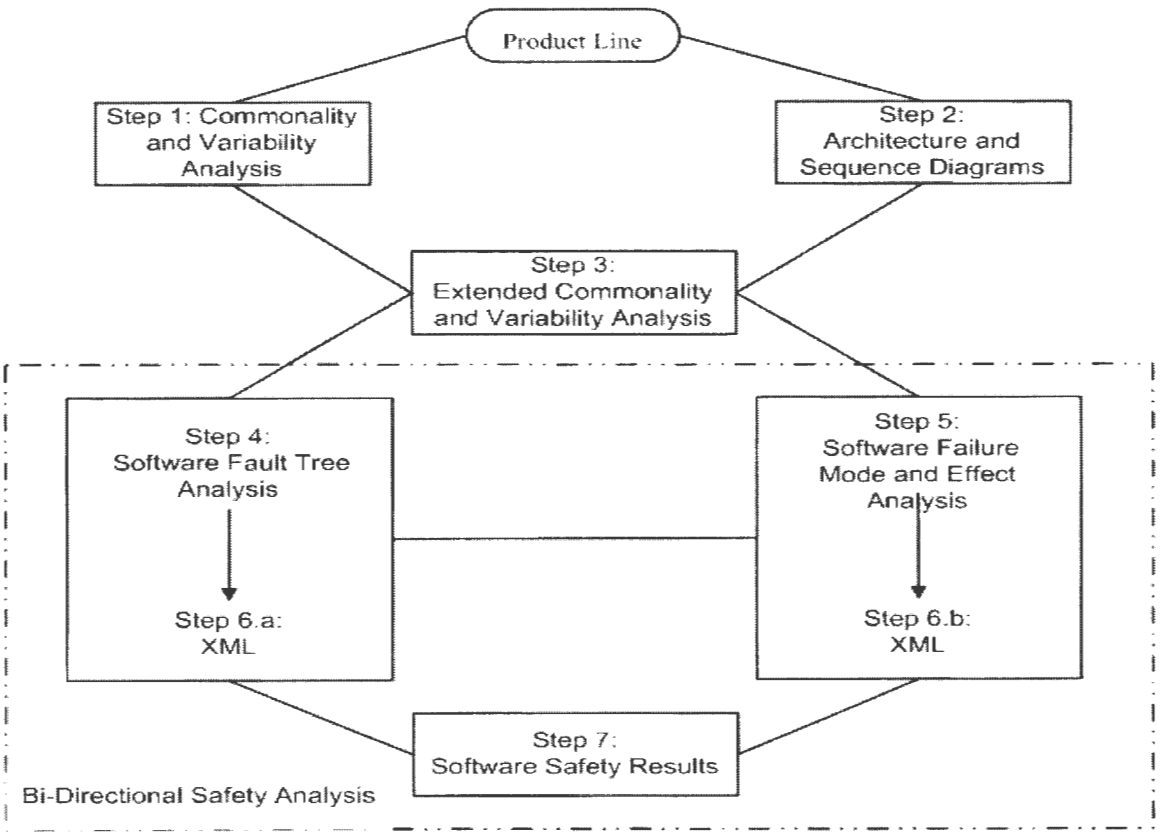

Figure 2. An Overview of the Safety Analysis Method

2 Related work

The work described here pulls together related work in two main areas: product line engineering and software safety.

In the area of product line engineering, our work draws on recent work in product line requirements and in product line architectures. We follow Ardis and Weiss, and Weiss and Lai in using a Commonality and Variability Analysis to identify and specify product line requirements [Ardis and Weiss, 1997; Weiss and Lai, 1999]. Section 3 describes our use of their approach in some detail. Our bi-directional analysis method is also compatible with alternative approaches to product

line requirements analysis, such as PuLSE [Bayer et al., 1999] and FORM [Kang et al., 1998]. However, because these approaches can enforce an ordering on the choices of features when a new product line member is built, we prefer the partial ordering of the Commonality and Variability Analysis.

Product line architecture is an extensively studied field. Bosch discusses how to develop the architectures to a product line and how to revise the product line architecture for different products. After analyzing all the products, an architectural design and its components are developed in three steps. The first step is to develop a product line architecture supporting the functional and quality requirements of the product line; the second step is to revise the product line architectural design to specific designs for different products; and the third step is to evolve the architectural design for new requirements and new products [Bosch, 2000]. Bass says a product line architectural design is built on the three points of "identifying variation", "supporting variation", and "evaluating for product line suitability". Product line architecture support for variation is represented by "inclusion or omission of elements", "inclusion of a different number of replicated elements", and "selection of versions of elements that have the same interface but different behavioral or quality attribute characteristics" [Bass, 2003]. Egyed points out a product line architecture provides "generic information common" to all the products in the product line and includes "a certain amount of ambiguity" in order to support variations in the individual product. "An individual architectural design is an instantiation of the product line architecture, which is less ambiguous" [Egyed et al., 2000]. The architecture for a product line is "a generic architecture from which the individual product architectures can be derived" and it provides two fundamental usages: one is the architecture for a whole product line can "capture the important aspects of the product line"; the other is an individual product's architecture can be instantiated from the product line architecture [Perry, 1998]. A "core" architecture or "baseline" architecture can be derived from taking the essential features of the product line [Lutz,

2000; Lutz and Gannod, 2003]. An advantage of the "core" product line architecture is that new products can be added to the product line as long as they meet the basic design constraints.

The use of UML, use cases, scenarios, and sequence diagrams has been widely studied for product lines. Bayer, e.g., uses scenarios to determine architectural requirements [Bayer et al., 2000]. And Clauß extends UML to support feature diagrams and variability in the standard kinds of UML diagrams [Clauß, 2001]. The KobrA method is a component-based development for product lines and aims at increasing the reuse of the product line. It describes how to use UML diagram model components for product lines [Atkinson et al., 2002]. John and Muthig describe how use cases can be applied for modeling the requirements for a system family and how a particular single-system use case approach can be extended to capture product line information and especially variability [John and Muthig, 2002].

Besides product line engineering, the other area of related work for our study was software safety, software safety analysis, investigation how software can jeopardize or contribute to the safety of a system [Leveson, 1995]. To date, there has been relatively little work directed specifically at the challenges of safety-analyses for product line. Initial work by Dehlinger and Lutz [Dehlinger and Lutz, 2004], and by Lu and Lutz [Lu and Lutz, 2002] have shown how software fault tree analysis, a successful safety analysis technology for single system, can be extended to product lines and reused, with caveats, for a new member of the product line. Progress has been made in ensuring that quality attributes are preserved in product line [Bass, 2003; Bosch, 2000].

Our work differs from previous work in two ways: (1) the focus to date has been on architectural analysis rather than on requirements analysis, as we do here, and (2) prior work has not clearly distinguished safety from other quality attributes such as performance, modifiability, safety, reliability, availability, and testability. Our research emphasizes the safety analysis as a unique property that must be assured across the product line.

3 Commonality and Variability Analysis

In this section, we describe Step 1 of the safety-analysis process summarized in Figure 1. This step uses the product-line Commonality and Variability Analysis to specify the requirements for the product line of interest.

Safety analysis of a product line begins with analysis of the requirements. A product line is a set of products that share common aspects and differ from each other through some variabilities [Ardis and Weiss, 1997; Weiss and Lai, 1999]. Through analysis of the requirements specification, we can identify software requirements related to safety and analyze them for completeness, consistency, and correctness.

The Commonality and Variability Analysis (CA) of a product line provides a requirements specification for the product line. A CA typically includes three parts: terminology, commonalties, and variabilities. The terminology is a "dictionary of standard terms"; the commonalities are "a list of assumptions that are true for all family members"; the variabilities "define how family members may vary" and "the scope of the family by predicting what decisions about family members are likely to change over the lifetime of the family" [Ardis and Weiss, 1997; Weiss amd Lai, 1999].

We demonstrate our methodology using the Door Control System for a Smart Home. The Door Control System is a safety-critical product line in that the software must function correctly to prevent intruders from entering and must respond correctly to life-threatening scenarios such as fires. A Smart Home system serves as an invisible housekeeper: it has sensors and agents to interact with humans and the environment to offer people convenience and safety. For example, the entrance doors can be opened only by inputting fingerprints or voiceprints [Cook, 2003; Youngblood, 2002; Smart-home project, Finland; Fellbaum and Hampicke]. We restrict our discussion in this paper to a Door Control System software product line with three products: a FrontDoor, a BedRoomDoor, and a

SecurityDoor.

This section provides examples from the Commonality and Variability Analysis for the Door Control System product line derived from the detailed descriptions of these three products. The CA consists of the terminology used, the commonalities, the variabilities, and the dependencies among the variabilities. The dependencies are constraints that the choice of one features places on the choices of other features [Doerr, 2002]. Note that we here exclude any non-behavioral commonalities and variabilities to focus on the software. The CA serves as a requirement specification for the product line and as an input to the product line's architectural design.

1.1 Terminology

Table 1

Terms for Commonality and Variability Analysis

Name	**Explanation**
Door alarms	The alarm of the door which will be triggered by the illegal entry of the door.
Registration	A person's ID (fingerprints or voiceprints) is input to the database to be recognized later. (Note that by "fingerprint registration", we mean the input of fingerprint images into a database. Some researchers instead use "registration" to refer to the "alignment" of a fingerprint image with stored images.)
Recognition	The door tests the ID to see if this person has access permission. If so, the door will open for this person, otherwise the door will not open.
Family member	A person who has access permission for the FrontDoor
Correct people/person	Person(s) who should be let in upon requesting a door's opening.
Wrong people/person	Person(s) who should not be let in upon requesting a door's opening.
Fire alarm	The alarm that indicates the fire
Forced entry	Perceived force from the outside environment to try to break the door or the lock, e. g., through impact or pressure on the door's surface above a limit.
Illegal input	Wrong ID.
Lock inside	If the door is locked inside, nobody can open the door from outside, even if this person belongs to the set of correct people.
People pass	A person's whole body passes the door from one side of the door to the other side.

3.2 Commonalities

C1. Accept family members' registration

C2. Recognize correct people

C3. Open door for correct people from outside

C4. Open door for people from inside

C5. Close door after people pass

C6. Sound door alarm upon forced entry

C7. Respond to the fire alarm

3.3 Variabilities

V1. Methods of recognition: fingerprint or voiceprint. FrontDoor: fingerprint; BedRoomDoor: voiceprint; SecurityDoor: fingerprint and voiceprint.

V2. Methods of registration: fingerprint or voiceprint. FrontDoor: fingerprint; BedRoomDoor: voiceprint; SecurityDoor: fingerprint and voiceprint.

V3. Whether or not the door can be locked inside.

V4. Open/close doors when fire alarm on (when BedRoomDoor door and SecurityDoor are closed while the fire alarm is on, it will be open by pushing).

V5. Methods of opening doors inside (weight, oral command, or input IDs).

V6. Methods of triggering doors' alarm: the FrontDoor's alarm can be triggered by the wrong fingerprint input three times; the SecurityDoor's alarm can be triggered by wrong IDs input once. The BedRoomDoor's alarm will not be triggered by the wrong fingerprint inputs.

3.4 Dependencies among variabilities

(1) The method of recognition must be the same as the method of registration.

(2) The door's response to the fire alarm is dependent on the method of recognition.

(3) Whether the door can be locked inside is dependent on the method of recognition.

(4) The method of opening the door from inside is dependent on the method of recognition.

(5) The ways to trigger the door's alarm is dependent on the method of recognition.

4 Extended Commonality and Variability Analysis Using Architecture Design and Sequence Diagrams

In this section, we describe Step 2 and Step 3 of Figure 2. These steps develop the architectural design and sequence diagrams, and use these to extend and refine the Commonality and Variability Analysis.

In order to capture the domain knowledge needed to perform product line safety analysis, information about design choices and constraints must be conjoined with the requirements specification. In particular, insight into how the system can "go wrong" and into the rationales behind the design choices is needed. The components in a system and the communication among them can be identified by the architectural design. Since a product line architecture is structured for reuse, its specification displays interactions and data transformation in term of handling of potential variations among the product line members. These architectural details provide a structure for assembling and evaluating the Software Failure Modes Effects Analysis and Software Fault Tree Analysis used in the safety analysis in Section 5. Since many safety-related scenarios involve particular sequencing of events of interactions, the safety analysis also needs a dynamic view of the system's execution. This is achieved by connecting a sequence-diagram perspective to the Commonality and Variability Analysis.

In this section, we introduce a core architecture for the product line and derive individual architectures for product line members. We extend the Commonality and Variability Analysis developed in Section 3 with information from the architectural design and from the scenarios' sequence diagrams. The resulting Extended Commonality and Variability Analysis (XCA) provides

the information needed to perform safety analysis on the product line. Furthermore, the XCA provides a foundation for reusing the safety analysis for new members of the product line in the future.

4.1 Architecture of DCS product line

The core architecture for a product line is a generic architecture, which not only captures the important common features of the product line, but also can be instantiated to be an individual product's architecture [Perry, 1998]. New products are added to the product line and reuse the architectural design, corresponding components, and corresponding safety analyses. Figure 3 shows a core architectural design for the Door Control System product line. The system is divided into two parts. One is the Central Control System; the other part includes the agents that communicate with the outside environment to detect changes and to provide required responses.

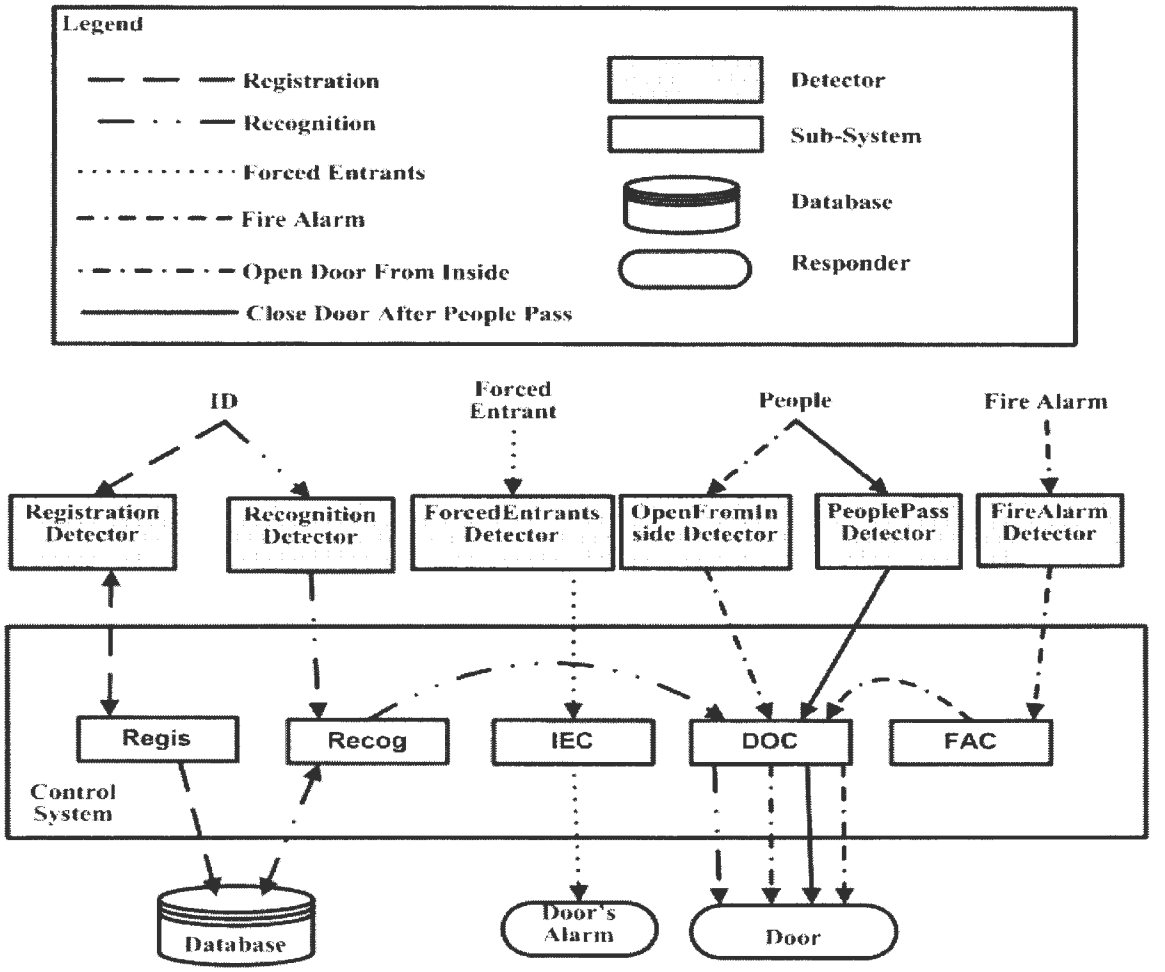

Figure. 3. The Core Architecture of DCS

The Central Control System has five components: RegistrationComponent(Regis), RecognitionComponent(Recog), IllegalEntryComponent(IEC), DoorOpenCloseComponent(DOC), and FireAlarmController(FAC). The detectors sensing the outside environmental inputs are RegistrationDetector, RecognitionDetector, OpenFromInsideDetector, ForcedEntryDetector, PeoplePassDetector, and FireAlarmSensors. The responders that affect the environment are the Door's alarm and the Door (the door's position and the lock status). There is also a database connected to the central Control System that stores the ID data.

Figure 4 is an individual architecture for the FrontDoor software system derived from the product line core architecture. The most obvious difference from the core architecture is that the sensors

have been instantiated to some specified detectors. For example, a "RegistrationDetector" in the core architecture is instantiated to an "F_registration detector" (Fingerprint) in the FrontDoor, a "V_ recognition detector" (Voiceprint) in the BedRoomDoor, and an "FV_registration detector" (both fingerprint and voiceprint) in the SecurityDoor. An "F_registration detector" is a camera to catch fingerprints; a "V_ recognition detector" is a sound detector to catch voiceprints; and an "FV_registration detector" is a detector that catches fingerprints and voiceprints at the same time.

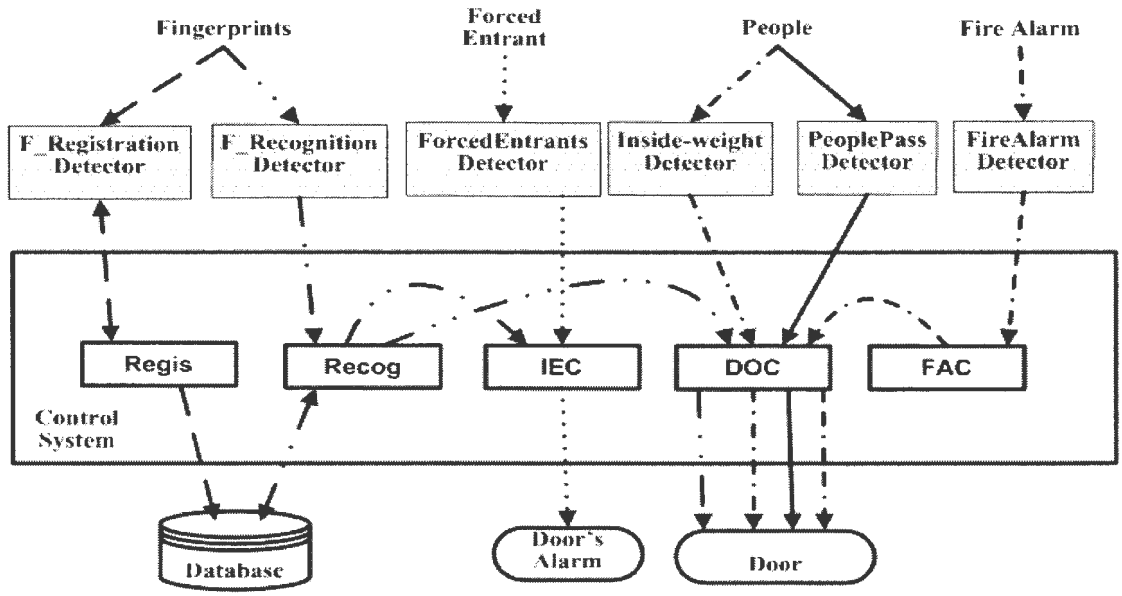

Figure. 4. Individual architecture for FrontDoor product

Another difference between the FrontDoor architecture and the core architecture is the connector in Figure 3 between the RecognitionComponent (Recog) and the IllegalEntryComponent (IEC), representing V6 of the variabilities in the Commonality Analysis. If there is an illegal ID input, the RecognitionComponent will send a signal to the IllegalEntryComponent, which will sound the door's alarm. Thus, the individual architecture for the FrontDoor requires an additional connector and some added functionality to generate and respond to the message that an illegal entry has been attempted.

4.2 Scenarios

Identification of the data flow and event sequencing is very important to the quality of the safety analysis, since our SFTA and SFMEA are built based on the communication of the components in the architecture diagrams. Scenarios are sequences of system activities used to illustrate system behaviors and "operational instances of system uses" by showing the interactions between the objects, including the messages or events flowing between the components [Allenby and Kelly, 2001; Goseva-Popstojanova et al., 2003; Sommerville, 2000]. Through analysis of scenarios and construction of the sequence diagrams, we not only obtain a deeper understanding of the product line, its requirements specification, architecture, and the interactions between components, but also identify all data transferred between components and all events.

For example, for the Door Control System product line, we derived the following seven use cases by associating every commonality and every variability not included in the commonalities with a use case.

1. *Registration*: registering the users' ID to the Door Control system (from Commonality C1)
2. *Entry*: entering the house from the outside (Including recognition from outside, opening the door, closing the door after the people pass, also including the illegal entrant handling) (from Commonalities C2, C3, C5, and C6)
3. *Exit*: exit the house from the inside (Including recognition from inside, opening the door, closing the door after the people pass, also including the illegal going out handling) (from Commonalities C4, C5, and C6)
4. *Fire alarm*: the door's response to the fire alarm (from Commonality C7)
5. *Bolt*: lock door from inside (from Variability V3)

Each use case has one main scenario and other miss-use scenarios. The scenarios are represented in sequence diagram where the top rectangles represent objects; the dashed lines connected to objects are the temporal "lifelines"; the arrows between the lifelines represent messages being sent between the objects; and the boxes on the dashed line are activations, showing the execution of a method in response to a message of that object. Figure 5 gives a portion of the sequence diagram for the Entry use case, showing the interactions between the components related to this use case and the messages transferred between them. This use case - Entry – to which this scenario belongs is safety-critical because there are two hazards are associated with it: - Not letting people out when they need to leave and Admitting intruders.

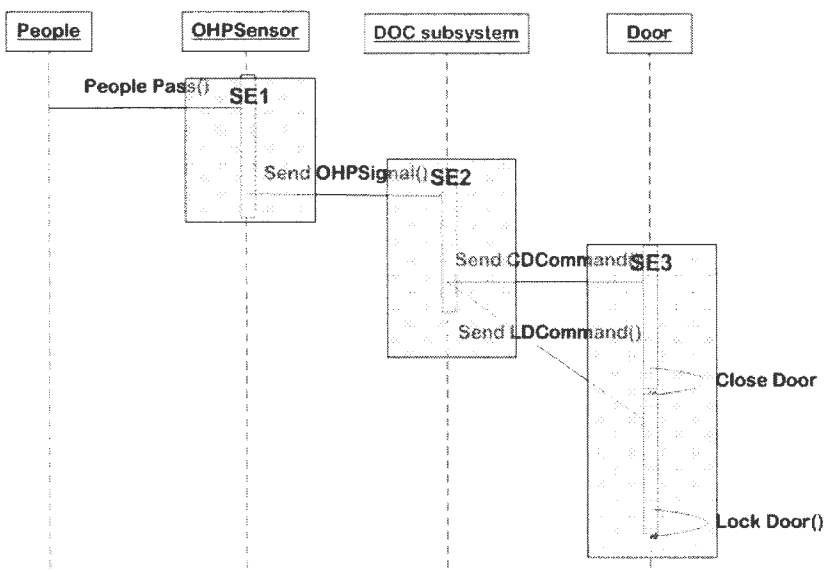

Figure. 5. Sequence diagram for the Close-Door-After-People-Pass scenario

Construction of the sequence diagrams not only made us get more understanding about the system but also identified a missing connector in the architecture diagram. The communication between F_registration detector and RecognitionComponent should be bi-directional, i.e., after the RegistrationComponent component has registered or rejected the ID, it should return a signal to the F_registration detector regarding success or failure.

The construction of the sequence diagram also led to the discovery of a need for additional

commands: the Lock Door Command (LDCommand) and the Unlock Door Command (ULDCommand). Initially only an Open Door Command (ODCommand) and a Close Door Command (CDCommand) were used to control the door's opening and closing. Analysis of the Fire Alarm use case showed that to close the door but keep it unlocked in the BedRoomDoor requires two pairs of commands. One command opens or closes the door; the other locks or unlocks the door. To close a door, the DCS needs to send out the CDCommand first, followed by the LDCommand; to open a door, the DCS needs to send out the ULDCommand first, followed by the ODCommand. When the fire alarm is on, the DoorOpenCloseComponent will send out the CDCommand and the ULDCommand to the BedRoomDoor and the SecurityDoor. However, in the FrontDoor, when the fire alarm is on, the DoorOpenCloseComponent will send out the ULDCommand and the ODCommand. The resulting corrections yielded a more accurate foundation for the subsequent Safety Analysis.

4.3 XCA

The Extended Commonality and Variability Analysis (XCA) derived from both the CA and the architecture and sequence diagrams provides the foundation of the product-line safety analysis.

The core architectural design and the events common to shared components give information about the commonalities for the product line, while the differences between individual architectures and the core, as well as the events that are particular to only some systems, give additional information about the variabilities. In the XCA we integrate the core architecture and common events with the commonalities of the product line. The variations of the product architectures from the core architecture and the non-common events are likewise associated with the variabilities of the product line.

First, we obtain events from the sequence diagrams. There are a total of five use cases, each represented by several scenarios. Each scenario is composed of several events. In a sequence

diagram, every connection between two actors or devices is an event, representing the sending of a message from a component and its receipt by another component. Every box on a vertical line indicates an internal event, usually the acceptance of a message and the generation of the next message within a component. For example, Figure 4 is a sequence diagram for the scenario in which the door closes after people pass, which is associated with three components: Object-has-passed sensor, DoorOpenCloseComponent (DOC), and Door. The connector between the sensor software and the DOC represents the event of sending a signal; the box in SE2 represents the internal events of generating the CloseDoorCommand (CDCommand) and the LockDoorCommand (LDCommand). For the safety analysis, we operate at a slightly abstract level by combining the receipt of a message, the resulting internal event, and the possible subsequent output of a command as one event. For example the SFMEA analysis considers three events in the sequence diagram in Figure 4 – SE1, SE2, and SE3.

Second, in order to maintain traceability between the events investigated in the safety analysis and the Commonality and Variability Analysis, each commonality in the CA is refined into several sub-commonalties associated with the corresponding events. For example, the commonality C5: "Close door after people pass" can be expanded into three sub-commonalties according to the three events described above. They are:

C5-1: "The event of a person passing through the door will trigger the activity of the Object-has-passed sensors, and the sensors software will send out the OHPSignal to the DOC";

C5-2: "After the DOC accepts the OHPSignal, the DOC will send out the CloseDoorCommand and the LockDoorCommand to the door"; and

C5-3: "After the door has accepted the CloseDoorCommand, it will be closed and after the door has got the LockDoorCommand, its lock will be locked".

C5-1 is a sub-commonality associated with the *component* Object-has-passed-sensors and the

event that the person has passed by. C5-2 is a sub-commonality associated with the component DOC and the event of accepting OHPSignal, generating CDCommand and LDCommand, and sending out them. C5-3 is a sub-commonality associated with the component Door and the event of accepting CDCommand and LDCommand.

In order to better manage the data, we group the sub-commonalties according to the use-cases to which they belong. For example, all the sub-commonalities expanded from C2: "recognize family members" and C3: "open door for family members from outside" are grouped together, since they are related to the same use case: "Entry".

Similarly, variabilities are refined into sub-variabilities. The method is slightly different from refining commonalities. For each variability we observe from the architectures and the sequence diagrams what the rationale is for the existence of that variability. For example, the variability V3 in the CA is "whether or not the door can be locked from inside". From examination of the individual architectures, we identified the components that allow the door to be locked in some products and not in others, i.e., the existence or non-existence of a door-lock button. From examination of the sequence diagrams, we found that the sequence diagrams that represent the variabilities have different components. In this example, we found that there are two sub-variabilities that determine the existence of this initial variability, V3. These are : V3-1: "whether or not there is a inside lock door button" and V3-2: "whether or not the DOC component can handle the ButtonPressedSignal".

Third, we construct the Extended Commonality and Variability Analysis (XCA) with the additional architecture and event information needed to support safety analysis. We provide below excerpts from the XCA of the Door Control System.

4.4 Terminology

Note that we group the terms according to the different use cases: Registration, Entry, Exit, Fire Alarm, and Bolt Door. Table 2 shows some terms that represent the data (commands and signals)

transferred between the components in two representative safety-related situations. The first is a safety-critical use case (Fire Alarm) and the second is a safety-critical scenario (Close Door After People Pass) of another use case (Entry) that also contains non-safety-critical scenarios.

Table 2

Data transferred between components

Data Name	**Data Description**
Situation one: Fire Alarm Use Case	
FAOnSignal	Fire Alarm-is-on signal
FARCommand	Fire Alarm-response command
CDCommand	Close door command
ULDCommand	Unlock door command
ODCommand	Open door command
Situation two: "Close door after people pass" scenario in Entry use case	
OHPSignal	Object-has-passed signal
CDCommand	Close door command
LDCommand	Lock door command

4.5 Commonalities

We here list the sub-commonalties for the two safety-critical situations listed above.

Use case 6: Fire alarm (C7: The door will respond to the fire alarm)

C7-1. The doors have a FireAlarmDetector sensor.

C7-2. The FireAlarmDetector will sense when the fire alarm is triggered and will send a signal to the FireAlarmComponent.

C7-3. After the FireAlarmComponent recerives the fire alarm signal, it sends the FireAlarmResponseCommand to the DoorOpenCloseComponent.

C7-4. The DoorOpenCloseComponent will send out the corresponding commands to the door after it has received the FireAlarmResponseCommand.

Use case 2: Entry (C5: Close door after people passing)

(Note that we here list only those commonalities related to the safety-critical scenario "Close door

after people pass")

C5-1. The door has a PeoplePassDetector sensor.

C5-2. The PeoplePassDetector will sense when people have cleared the door and then will send out the ObjectHasPassedSignal to the DoorOpenCloseComponent.

C5-3. After the DoorOpenCloseComponent has accepted the ObjectHasPassedSignal, it will send out the CloseDoorCommand and the LockDoorCommand, respectively and continuously to the door after it gets the ObjectHasPassedSignal.

C5-4. After the door has received the CloseDoorCommand it will be closed and after the door has received the LockDoorCommand its lock will be locked.

4.6 Variabilities

We list all the variabilities.

V1-1. The RegistrationDetector is different in different products: F_registration detector (fingerprints) or V_registration detector (voiceprints) or FV_registration detector (both)

V2-1. The RecognitionDetector is different in different products: F_ recognition detector or V_ recognition detector or FV_ recognition detector.

V3-1. Whether or not there is an inside lock door button

V3-2. Whether or not the DOC can handle the ButtonPressedSignal

V4-1. Whether or not the DOC sends out CloseDoorCommand and UnlockDoorCommand, or UnlockDoorCommand and OpenDoorCommand, to the door after it gets FireAlarmResponseCommand

V4-2. Whether or not the door will be open or closed after the fire alarm is on.

V5-1. The OpenFromInsideDetector is different in different products: InsideWeightDetector or InsideVoiceDetector.

V6-1. Whether or not the RecognitionComponent will count the number of illegal IDs that have

been input and send out the IllegalInputsignal.

V6-2. Whether or not the IllegalEntryComponent will accept the IllegalInputSignal form RecognitionComponent component.

4.7 Tag the commonalities and variabilities to the architecture diagram

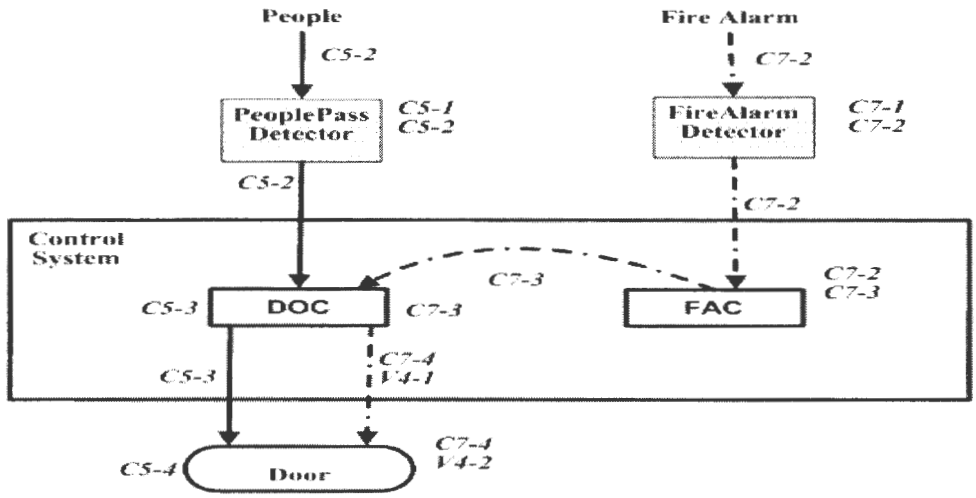

Figure. 6. Architecture of FrontDoor tagged with commonalities and variabilities

The XCA gives us a clear requirements specification of every component in the DCS. We see that even in this simple product line with only a few variabilities, that the variabilities impose architectural constraints. To verify the completeness of the XCA, we label the connectors in the architecture diagrams with the indices of the associated commonalities and variabilities. Each commonality and variability should associate with one or many connectors in the architecture diagrams. At the same time, every connector in the architecture diagrams should be tagged with one or many commonalities or variabilities. Figure 6 shows an architectural design of the FrontDoor tagged with the commonalities and the variabilities from the safety-critical situation: the Fire Alarm use case and the Close door after people pass scenario.

By associating the components and connectors in the architecture diagrams with the commonalities and variabilities that use them, we capture additional information needed for the

safety analysis and establish the architecture as a structuring device to generate the SFTA and SFMEA.

5 Software Safety Analysis

In the previous section, we described Step 2 and Step 3 of the product-line safety-analysis process shown in Figure 1. In this section, we discuss Steps 4-6 of this process, namely the production of the safety-analysis artifacts SFTA and SFMEA, and the translation of both into XML.

The method for software safety analysis of a product line proposed here uses the XCA and a hazards list to drive the bi-directional safety analysis. In this section, we first describe the hazard analysis and the backward and forward safety analyses. We then describe their translation into XML files that serve as reusable assets for the product line. Finally we describe the results of applying our method to the Door Control System.

5.1 Hazard Analysis

A detailed discussion of the hazard analysis is beyond the scope of this paper. Briefly, we used a Functional Hazard Assessment (FHA) to identify the following eight hazards [Allenby and Kelly, 2001]:

1) Don't let in the correct people.
2) Let in the wrong people.
3) Don't let people out.
4) When the fire alarm is on, the door doesn't response as it should and people inside cannot get out.
5) The door cannot be locked from the inside, so people inside don't have privacy.
6) The door cannot be opened after it is locked inside, so people cannot get out or get in.

5.2 Backward Safety Analysis – Software Fault Tree Analysis (SFTA)

SFTA is a "top-down" safety analysis technique that is used to identify the errors, faults and failures that could contribute to hazards. It is a means for analyzing causes of hazards, which are identified as the top events. A backward search from the top event is performed to find out combinations of contributing causes of the hazards [Leveson, 1995; Lutz and Woodhouse, 1997]. The tree is equivalent to a predicate statement with the nodes connected by AND and OR gates. The SFTA is guided by the XCA.

Every intermediate node is decomposed into sub-trees. The sub-trees contribute to the occurrence of the parent. Combination errors include the errors caused by the occurrence of two or more correct events at the same time, errors caused by the race conditions of two or more events, errors caused by the occurrences of two or more errors at the same time, and errors caused by the handling of two or more errors at the same time. Every error is decomposed into sub-trees until the leaf nodes are reached. The leaf nodes are analyzed using the failure modes from SFMEA, such as "absent data", "incorrect data", "wrong timing of data", "duplicate data", "halt/abnormal termination of event", "omission", "incorrect logic/event", and "timing/order", [Lutz and Woodhouse, 1997]. We considered if those failure modes of the data and events that are associated with each component could contribute to the errors of the component that were on the bottom of the fault tree.

We catalog the fault tree's nodes into five types: "Hazard", "Intermediate", "Leaf", "ProductsDivision", and "Reused". The root node is level zero; a hazard node is at level one of the fault tree. An intermediate node is a node that is not a leaf, a hazard, a reused node, or a ProductsDivision node. A leaf node is a node that is at the bottom level of the fault tree. A reused node is a shorthand specification of a sub-tree that repeatedly occurs. A ProductsDivision node is a node related to the divisions of the tree associated with different products. ProductsDivision nodes provide an abstraction of the concrete errors of different products in the nodes of the next level. The

advantages of these nodes are the following:

- Understandability. With these kinds of nodes, we can easily locate the place that the variations occur.
- Reuse and evolution. The sub-tree will be a complete tree for several products with the same variability. In this way, we can easily edit and trim the fault tree [Dehlinger and Lutz, 2004].

Since every node in the fault tree can be connected to a product or a group (subfamily) of products, we tag every node in the fault tree with the appropriate indices of the XCA, so that every node in the fault tree is associated with at least one commonality or one variability. The reason for the tagging is to make the product line's fault tree easy to understand and easy to edit and prune in the future. For example, suppose we want to derive a new product with the additional privacy feature of being able to lock the FrontDoor from inside against even people with valid access. In this case, we can check the nodes' tags to identify all the nodes marked with the tags of the commonalities and variabilities related to Lock-Door-Inside, Fingerprint-Registration and Fingerprint-Recognition.

We have created an entire fault tree for the Door Control System product line. The root node is "Malfunction of the Door Control System" and the nodes in level one are the six hazards described above. The nodes in level two are the errors and faults that can contribute to the hazards, etc. We omit the fault tree here for space reasons but include pieces of its XML representation below.

5.3 Forward Safety Analysis – Software Failure Mode and Effect Analysis

The Software Failure Mode and Effect Analysis (SFMEA) is a bottom-up forward search from failure modes associated with data and events to the effects that are caused by those failure modes. [Leveson, 1995; Lutz and Woodhouse, 1997]. The first step is to identify all the components in the system, which we have already done through the architecture diagrams. Lutz and Woodhouse [Lutz

and Woodhouse, 1997] provide a list of generic failure modes: four associated with data communication and four associated with event processing. The four failure modes for data are "incorrect value", "absent value", "wrong timing", and "duplicated value". The four failure modes for events are "halt/abnormal termination", "omission", "incorrect logic/event", and "timing/order". The events are obtained from the sequence diagrams as described in Section 4.2. The data are obtained from the architecture diagrams and the sequence diagrams. In the architecture diagrams, the communication data is the data transferred along the connectors. In the sequence diagrams, the data are transferred between two vertical lines

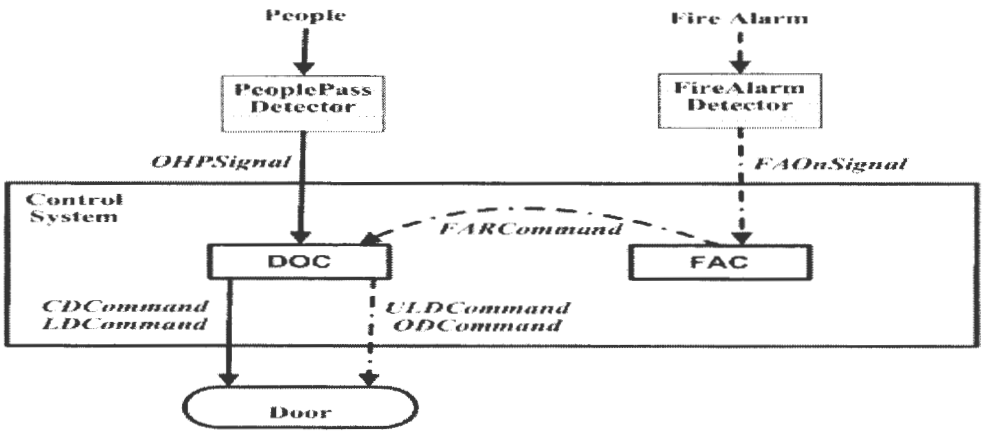

Figure. 7. Mapping data to the architectural design

The safety analysis groups the data according to the use cases. For example, in the Fire Alarm use case group, the data include FireAlarmOnSignal, FireAlarmResponseCommand, CloseDoorCommand, UnlockDoorCommand, and OpenDoorCommand. Each data item is essential to make this use case occur successfully and no other data is necessary to its occurrence.

Figure 7 shows the relationship between the use case data and the architecture for the use case: Fire Alarm use case and the portion of the Entry use case: Close-Door-After-People-Pass scenario. By tagging the SFMEA's data with the architectural connectors, we ensure coverage of the data that communicate among components. That is, the SFMEA depends on the architecture diagrams. However, note that different use cases may use the same data. For example, the CloseDoorCommand

is in both the Fire Alarm use case and in the Entry use case. Since both the reasons for the generation of these data and the hazards to which they can contribute may differ, we treat data with the same name but from different use cases as distinct from each other.

Table 3

Schema of SFMEA's data table

Data Item	Group	Product	Data Failure Mode	Local Effect	End Effect	Possible Hazard
The name of the datum.	The name of the group that the datum belongs to.	The products' name that the datum is associated.	Incorrect value, absent value, wrong time, or duplicated value.	The effect that occurs to the components that the datum is directly associated with if the datum is in the failure mode.	The system failure if the local effect occurs.	The possible hazard that happens if the end effect occurs.

Table 4

SFMEA on the OpenDoorCommand of Fire Alarm group

Data Item	Group	Product	Data Fault type	Local Effect	End Effect	Possible Hazard
OpenDoorCommand	Fire Alarm	FrontDoor	Incorrect Value	The DoorOpenClose component does not receive FireAlarmResponseCommand, but the DoorOpenClose sends out OpenDoorCommand.	When the fire alarm is not on, the door is incorrectly opened.	Lets in the wrong people.
			Absent Value	The DoorOpenClose component receives FireAlarmResponseCommand, but the DoorOpenClose does not send out OpenDoorCommand.	When the fire alarm is on, the door is incorrectly not opened.	The fire alarm is on and people cannot exit.
			Wrong timing	The DoorOpenClose component receives the FireAlarmResponseCommand, but the OpenDoorCommand is delayed for at least 1 min.	When the fire alarm is on, the door's opening has been delayed for at least for 1 min.	The fire alarm is on and people cannot exit quickly.
			Duplicate Value	The DoorOpenClose component receives the FireAlarmResponseCommand, and then the DoorOpenClose component sends out two OpenDoorCommand.	When the fire alarm is not on, the door is incorrectly opened.	Let in the wrong people.

Table 5

SFMEA on an event for the FrontDoor product in the Fire Alarm use case

Group	Product	Event Fault type	Local Effect	End Effect	Possible Hazard
Fire Alarm	FrontDoor	Halt/Abnormal termination	The DoorOpenCloseComponent receives the FireAlarmResponseCommand, but it does not send out the UnlockDoorCommand and the OpenDoorCommand to the door.	The fire alarm is on, but the door is incorrectly not opened.	The fire alarm is on and the people cannot exit.
		Omission	The DoorOpenCloseComponent receives the FireAlarmResponseCommand, but it does not send out the UnlockDoorCommand and the OpenDoorCommand to the door.	When the fire alarm is on, the door is incorrectly not opened.	The fire alarm is on and the people cannot exit.
		Incorrect logic/event	The DoorOpenCloseComponent does not receive FireAlarmResponseCommand. But it sends out the OpenDoorCommand or UnlockDoorCommand or both.	When the fire alarm is not on, the door is incorrectly opened.	May let the wrong people in.
		Timing/Order	The DOC component receives the FireAlarmResponseCommand, but sends the UnlockDoorCommand later than the OpenDoorCommand.	The fire alarm is on. The door is unlocked, but it's incorrectly not opened.	When the fire alarm is on, people cannot exit.

As shown in Table 3 the schema of our SFMEA data table includes "Index", "Data Item", "Group", "Product", "Data Failure Mode", "Description", "Local Effect", "End Effect", and "Possible Hazard".

Note that in every row in the data table, a failure mode of a datum is identified and we assume that every other datum is not in any failure mode. In reality, this is not always true. For example, a failure of the OpenDoorCommand is probably always associated with a failure of the UnlockDoorCommand. However, the SFMEA's event table does consider such combinations of errors, as long as these data are in the same event.

Table 4 is excerpted from the SFMEA for the OpenDoorCommand in the Fire Alarm use case. In our example, we only include one failure effect for each failure mode, although there are several effects associated with some failure modes.

The schema of the event table of SFMEA is similar to the data table's. Table 5 shows a SFMEA on an event for the FrontDoor product in the Fire Alarm use case. In this event, the DoorOpenCloseComponent receives the FireAlarmResponseCommand, and then generates and sends an UnlockDoorCommand and an OpenDoorCommand to the Door. Again, we here show only one failure effect per failure mode.

As in the data table, although the events table may have two or more events with different names doing the same thing, we still name them differently because their triggers and potential hazards may be different.

5.4 Convert SFTA and SFMEA to XML files

In order to make the safety analyses available and reusable when adding new products to the product line, we convert the results of the SFTA and the SFMEA into XML files in a partially-automated step. The reasons for selecting XML in this research (e.g., instead of a relational database) are as follows:

Basically, the advantage of XML is that since the output of the SFTA is a tree, it can be readily translated into XML by the FaultCat tool. Furthermore, Finally, XML is as easy to manipulate as a relational database and as powerful for our purposes.

1. The result of the SFTA is a fault tree, which is stored as an XML file by the FaultCat tool used to construct the fault tree.
2. The representation of fault trees in XML is becoming fairly standard. For example, the Galileo fault tree tool at the University of Virginia uses XML [Sabanosh and Sullivan, 2001], as do commercial fault-tree tools, such as Relex. XML will likely become even more widely accepted

in industry as the exchange language for analysis toolsets as XML parsers are built into programming languages.

3. XML representations of fault trees are easy to understand and manipulate, so appeal to software developers. For example, the correspondence between the graphical representation of nodes and their hierarchy in the XML nodes is clear without training.

The resulting XML files also allow automated comparison of the two safety analyses. In addition, a product line can be expected to evolve over time. The XML representation can be easily edited to incorporate changes.

For example, to build a fault tree for just the product BedRoomDoor, we can partially-automatically extract those nodes belonging to the BedRoomDoor to build a new fault tree for that product. In this way, the reuse of safety analysis of a product line can be achieved more accurately and quickly.

```
<!ELEMENT fault-tree ( node+ ) >
<!ELEMENT node (Name, Parent, Gate, Type, Products, ComOrVar, Description, Children) >
<!ELEMENT Products (Product +) >
<!ELEMENT Description (Content, KeywordsSet) >
<!ELEMENT KeywordsSet (Keyword +) >
<!ELEMENT Children (ChildrenNum, Child +) >
<!ELEMENT Gate ( "" | "Or"| "And" ) >
<!ELEMENT Type ( "Hazard" | "Error" | "ProductsDivision" | "Reused" ) >
```

Figure. 8. Wanted XML DTD elements

We construct the fault tree using the Fault Tree Creation & Analysis Tool (FaultCAT) [Burgess, 2003], which allows us to draw and edit the fault tree and convert it into an XML file. Because we want every node in the fault tree to be an independent node (somewhat similar to [Sabanosh and Sullivan, 2001]), we wrote a Java program, which takes as input the original XML file with the format from FaultCAT and writes out an XML file with the desired format as shown in Figure 9 in DTD format [DTD tutorial]. The advantage of our format is that it is easy to understand and easily

program readable. Figure 8 is an example of a leaf node of the Fault Tree in our format.

```
<!ELEMENT SFMEA_data ( Data + ) >
<!ELEMENT Data (NodeIndex, DataIndex, DataName, DataNameLong, Products, Errors)>
<!ELEMENT Errors (OneError +) >
<!ELEMENT OneError (OneErrorIndex, DataFailureMode, Description, LocalEffect, EndEffect,
PossibleHazard)>
<!ELEMENT LocalEffect (Reused | (LocalEffectContent, KeywordsSet) >
<!ELEMENT EndEffect (Reused | (EndEffectContent, KeywordsSet) ) >
<!ELEMENT KeywordsSet (Keyword+) >
```

Figure 9. DTD for SFMEA

```
<Node>
        <Parent>B4</Parent>
        <Name>B38</Name>
        <Type>Error</Type>
        <Gate>Or</Gate>
        <ComOrVar>
            <Name>Com</Name>
            <Products>
                <Product>P_F</Product>
                <Product>P_B</Product>
            </Products>
        </ComOrVar>
        <Description>
            <Content>The DOCSub does not accept OHPSignal. But it sends out the ODCommand.
             So the door is closed before the correct people's passing
            </Content>
            <KeywordSet>
                <Keyword>DOCSub</Keyword>
                <Keyword>does not</Keyword>
                <Keyword>accept</Keyword>
                <Keyword>OHPSignal</Keyword>
                <Keyword>CONDITION</Keyword>
                <Keyword>DOCSub</Keyword>
                <Keyword>sends out</Keyword>
                <Keyword>ODCommand</Keyword>
            </KeywordSet>
        </Description>
        <Children>
            <ChildrenNum>0</ChildrenNum>
        </Children>
</Node>
```

Figure 10. A node of the SFTA

For the SFMEA table each data or event is converted into one node of the XML file. Figure 10

gives a description of the XML file of the SFMEA's data table, using DTD [DTD tutorial]. We wrote a Java program to transfer the tables of the SFMEA to a XML file using the Java Excel API.

By partially automated, we mean that the Java programs to translate the XML files and the Xlinkit rules to compare the XML files only have to be written once. Subsequently, the programs automatically compare and check the files. The advantage of having partially automated comparison is that it is significantly faster and lower-cost than manual comparison.

6 Results and Evaluation

This section describes Step 7 of the safety-analysis process for product lines that was summarized in Figure 1. The section presents and evaluates the results both in terms of the automated consistency checking of the forward vs. backward safety analysis results and by discussing the missing software safety requirements for the product line found by this method.

6.1 Consistency Checking of SFTA and SFMEA

To evaluate the consistency and completeness of the safety analyses, we use partial automation to compare the SFTA and SFMEA XML files. The tool that we used to do the comparison of the keywords set, called "Xlinkit", can apply rules on multiple XML documents [Nentwich, 2002; Nentwich et al., 2002]. To accomplish this we conducted a comparison of SFTA and SFMEA, i.e., comparing every node of the fault tree with every failure effect of the SFMEA and vice versa. The method of comparison is using the keywords set and Xlinkit. Figure 11 shows a portion of the code for the Xlinkit rule that checks if the fault tree's nodes are in the SFMEA's effects. For every node in the fault tree for which there exists the same failure effect in the SFMEA data table, the node's keywords must exist in the keywords of that failure mode's failure effect and vice versa.

A keyword of a phrase is a word that is necessary to express the correct meaning of this phrase. So, to compare two phrases, we can compare two keyword sets. Our comparison of the SFTA and

the SFMEA was based on the assumption that the architectural design used in SFTA and the one used in SFMEA are at the same level of detail. More precisely, if the architectural design used in SFTA has a component A, then the architectural design used in SFMEA has the same component A; if this component A has n number of sub-components in the design used in SFTA, then the component A has the same number of sub-components in the design used in SFMEA.

```
<or>
   <exists var="oneDataError" in="/SFMEAData/Data/Errors/OneError">
      <or>
         <and>
            <forall var="oneSFTKeyword" in="$oneNode/Description/KeywordSet/Keyword">
               <exists var="oneDataLocalKeyword"
                in="$oneDataError/LocalEffect/KeywordsSet/Keyword">
                  <equal op1="$oneDataLocalKeyword/text()" op2="$oneSFTKeyword/text()" />
               </exists>
            </forall>
            <forall var="oneDataLocalKeyword"
             in="$oneDataError/LocalEffect/KeywordsSet/Keyword">
               <exists var="oneSFTKeyword" in="$oneNode/Description/KeywordSet/Keyword">
                  <equal op1="$oneDataLocalKeyword/text()" op2="$oneSFTKeyword/text()" />
               </exists>
            </forall>
         </and>
         <and>
         ......
         </and>
      </or>
   </exists>
</or>
```

Figure. 11. A part of a rule of Xlinkit

We compared the *faults* of SFTA and the *failure effects* from SFMEA from the Fire Alarm use case and the "Close-Door-After-People-Pass" safety-critical scenario from the Entry use case. There were thirty-eight nodes in the SFTA's Fire Alarm use case and ten nodes in the SFTA's Close-Door-After-People-Pass scenario. In the corresponding SFMEAs, there were eight data items analyzed and thirty-two failure effects; nine events analyzed and seventy-five failure effects. Note that between

the data tables and the event tables, there were many redundant failure effects, i.e., some failure effects appeared repeatedly. Within failure modes, a small set was identical and some had the same meaning.

From the comparison using Xlinkit, we found that the SFMEA is more complete than the SFTA when both of them were at the same level of detail. Here, the same level of detail refers to the architecture level as described above. Seventeen nodes in the fault tree with the type "Error" and with the gate "Or" or "And", were not in the SFMEA data table or in the SFMEA event table. After checking the seventeen nodes manually, we found that, in fact, fifteen nodes do have corresponding effects in the SFMEA. One reason that Xlinkit could not match them is because they had different keywords. For example, node B11 in the SFTA was not found in the SFMEA. The KeywordSet of the B11 node in the fault tree included the keywords: "door", "does not", "accept", "CDCommand", "door", "close", "without", "people", and "pass". However, after checking manually, we found a failure mode in the SFMEA with the KeywordSet containing "door", "does not", "accept", "CDCommand", "ULDCommand", "door", "close", and "unlock". Although these two KeywordSets were different, they actually described the same fault. Both relate to the undesirable behavior of the door closing automatically without having been commanded to close. Another reason that Xlinkit could not match some KeywordSets is that in one case, the analyst had inadvertertently omitted a pre-condition from the SFTA but included it in the SFMEA.

We also checked whether the failure effects from the SFMEA matched the faults of the SFTA. Again we found a significant number of initial mismatches (sixteen failure effects in the event tables that were not found in the SFTA and twelve failure effects in the data table that were not found in the SFTA, excluding timing errors). We manually checked to see why those effects were not in the SFTA. Half of the mismatches were due to the keywords sets' variability. The others were due to the SFMEA being more detailed than the SFTA. For example, only the SFMEA considered the

possibility that LockDoorCommand is sent while the CloseDoorCommand is not when the door is closed. The SFTA considers the two commands together.

6.2 Identification of New Safety Requirements

The application of the safety analysis methodology to the Door Control System product line identified some missing safety requirements. Some new requirements were commonalities that addressed shared hazards. Other new requirements affected only some products in the product line (i.e., were variabilities).

An example of a new shared safety requirement came from the forward search that showed that if the digitalized fingerprint image data were in the failure mode "incorrect value", this could have the effect of letting in intruders. Performing a backward search using the failure mode "incorrect value" as the root to find out why it failed to give the correct fingerprint images identified the possible cause that the F_ recognition detector does not accept the new input and continues to send out the old data. This finding resulted in a new safety-related requirement that sensors must purge old data, thus also requiring the addition of an expiration time for all data that is used in a control decision.

We also found a new safety requirement by inspection of the architecture for race conditions. In every product, if a person has passed the door, the Object-has-passed sensors will send out the ObjectHasPassedSignal to the DoorOpenClose component, which will then send out a CloseDoorCommand and an LockDoorCommand, respectively. This is a correct operation of the Door Control System. However, in the FrontDoor product, when the fire alarm is on, the FireAlarmDetector will send out the FireAlarmOnSignal to the FireAlarmComponent, which will send out the FireAlarmResponseCommand to the DoorOpenCloseComponent. The DoorOpenCloseComponent will then send out an UnlockDoorCommand and an OpenDoorCommand to the door.

A race condition can occur if at the moment that a person passes through the door, the fire alarm

is on. If the UnlockDoorCommand and OpenDoorCommand get to the door later, then there is no hazard. However, if the CloseDoorCommand and the LockDoorCommand get to the door later, then the door will be closed and locked while the fire alarm is on. In the product BedRoomDoor, the same race condition can also cause this dangerous behavior when the fire alarm is on.

To avoid this race condition, a new safety-related behavioral commonality was added to the system: after the DoorOpenCloseComponent receives the FireAlarmResponseCommand, it will not accept any other commands until the fire alarm is off. This also results in a new requirement for additional functionality to move the DoorOpenCloseComponent out of the frozen state.

Other instances of incompleteness in the product line requirements found by the bi-directional safety analyses were that there should be sensors on the edges of the doors to prevent users from being pinned and that there should be software time-outs to control the door's opening and closing. We also found a missing variability for the BedRoomDoor and the SecurityDoor, i.e., that when the fire alarm is on, the door must be unlocked and closed, even if the door's initial state is locked inside. Through the safety analysis (specifically the SFMEA) we also discovered a new possible hazard: "people are pinned between the doors," common to all the products in the Door Control System product line.

7 Summary

This paper has presented a method for performing safety analysis on a software product line and demonstrated the method on three members of a safety-critical product-line, the Door Control System for a SafeHome application subsystem. The work described here extended the product-line commonality and variability analysis with domain information from the product-line architecture and sequence diagrams. The resulting representation, called the Extended Commonality Analysis, was then used to guide the bi-directional safety analysis. The intermediate products of the bi-directional

safety analysis, SFTA and SFMEA, were converted to XML files and, using rules coded by the analyst, automatically compared via the software package, xlinkit. Omissions and inconsistencies were then identified and removed, providing a more-thorough safety analysis. Making the safety analyses available to the projects as XML files provides an important first step toward partially automated updating of safety analyses as requirements evolve, and toward reuse of the safety analyses in the application-engineering phase as new systems are built.

Findings from application of the bi-directional safety-analysis method included new safety-related software requirements both for all the systems in the product line (commonalities) and for only some of the product-line systems (variabilities), as well as discovery of a new hazard, that people can be pinned by the door. The paper provides a structured method and step-by-step guidelines for deriving a safety analysis from an extended commonality analysis in order to improve the safety of the software product line. The method is general, so can help assure the safety of other critical product lines. To this end, we are currently investigating the scalability and domain-independence of the method in an application to a real-world industrial product line.

CHAPTER 3 SECURITY REQUIREMENTS ANALYSIS FOR BUNDLE PROTOCOL IN DELAY TOLERANT NETWORK

Abstract

Delay Tolerant Network protocols support communication in deep-space environments where long signal delay and frequent service interruptions cause standard network protocols to be inadequate. A Delay Tolerant Network is potentially susceptible to denial-of-service attacks due to the highly variable round-trip communication time, the rarity of the being-able-to-communicate period among nodes, the scarcity of memory in nodes, and their lacking computing resources. The contribution of this paper is a systematic analysis of the security requirement for the Bundle Protocol, the network layer interposed between the transport layer and the router applications to handle delays. Our approach extends an existing framework for security requirement with a bi-directional analysis in order to methodically investigate the correctness and completeness of the security requirements of the protocol. Results identify four additional requirements to prevent message flooding attacks that would cause a denial of service in Delay Tolerant Networks.

1 Overview

A Delay Tolerant Network (DTN) is an end-to-end network providing communications in extreme environments such as deep space communication or sensor-based networks. Such network experience extremely long signal propagation delays, on the order of seconds, minutes, or hours rather than milliseconds, frequent and lengthy interruptions in connectivity, low levels of traffic coupled with high rates of transmission error, meager bandwidth and highly asymmetrical data rates. Existing protocols for today's Internet are not sufficient to fulfill the requirements associated with the highly stressed environments [Cert et al., 2005; Scott, 2005; Symington, 2005; Burlegigh, 2005; Ramada, 2005; Fall, 2005; Warthman, 2003].

The Delay Tolerant network (DTN) architecture was proposed to solve the problems described above and to embrace the concepts of occasionally-connected networks that may suffer from frequent partitions and that may be comprised of more than one divergent set of network communication protocol families. Research on DTN has produced the proposals, the specifications, and the implementations of the Bundle Protocol and the LickLider Protocol [Cert et al., 2005; Scott, 2005; Symington, 2005; Burlegigh, 2005; Ramada, 2005, Fall, 2005; Warthman, 2003]. Although both protocols address security issues, they have not to date been thoroughly analyzed for completeness and correctness. Because of the importance of the role that security is playing in DTN [Ramada, 2005], we believe that this kind of analysis should be done as early as possible.

The security requirements analysis approach that we use in this research is based on the Framework of Core Security Requirements Artefacts approach (FCSRA) proposed by Moffett, Haley, and Nuseibeh [Moffett et al., 2004]. This approach shown in Section 3 integrates the requirements engineering and security analysis in viewing security requirements as constraints on the functional requirements needed to fulfill security goals [Moffett et al., 2004]. It utilizes Jackson's Problem Frame to analyze system functional requirements and security goals and develops security

requirements specifications in the form of a Problem Frame [Jackson, 2001; Moffett et al., 2004]. The Problem Frame illustrates the software requirements through specifying the machine (software)'s behavior and interactions between the machine and the other domains (environment components) and provides a means of analyzing and decomposing requirements [Jackson, 2001; Moffett et al., 2004; Harley, 2004]. We introduce the FCSRA in Section 2.

The benefits of using FCSRA are five-fold: 1) Problem Frame focuses only on requirements. We are only interested in the requirements and do not want to bring the architectural design into our analysis; 2) Easy to obtain the traceability of security requirements; 3) Easy to maintain with rapid evolution expected; 4) Support reuse; and 5) Easy to apply and understand. Each of these benefits is described further in Section 5.

However, the Framework of Core Security Requirements Artefacts approach (FCSRA) has the weakness that it does not have a convincing method to systematically prove the completeness and the correctness of the resulting security requirements specifications. Instead it tries to argue using a "correctness argument" to convince others that the proposed machine will ensure the requirements [Jackson 2001; Moffett et al., 2004; Harley, 2005]. To address this weakness, we use an extended bi-directional analysis to assess the correctness of the result from the FCSRA approach.

The bi-directional analysis combines a forward search from potential failure modes to their effects with a backward search from feasible hazards to the contributing causes of each hazard. The combination of the forward and backward search has proven effective in discovering latent safety requirements [Lutz and Woodhouse, 1997].

In the work, instead of using the standard artifacts of a domain engineering process, such as system architecture, use cases, and scenarios, the extended bi-directional analysis method takes the Problem Frame and the behavior descriptions of the machine as input. Through the application of the extended bi-directional analysis we provide a structure to methodically investigate the correctness of

the security requirements specification. This also maximizes the coverage of the problem described in the Problem Frame.

This work focuses on the security requirements to prevent denial-of-services attacks, especially message flooding. The extended bi-directional analysis method takes the result from the Problem Frame and discovered that the denial-of-services will happen if the trust assumptions are violated. A trust assumption is a decision about how much and how to trust the properties of domains that make up the system [Jackson, 2001; Haley 2004]. An example of a trust assumption is that the DTN users shall not be compromised and used maliciously to inject too many messages into the network.

As the result, we propose four security requirements shown in Section 4.4 to prevent the denial-of-services caused by compromised satellites or clients. They strengthen the correctness arguments of the Problem Frame by weakening the dependence on the required trust assumptions. Finally the approach iterates back to the Problem Frame and updates the artifacts using the new security requirements.

The rest of the paper is organized as follows. Section 2 presents the background knowledge used in the paper: Framework of Core Security Requirements Artefacts, and Delay Tolerant Network. Section 3 walks through the process of security requirements analysis. Section 4 describes the results of this approach. Section 5 provides a discussion of the research experience in applying this approach. The last section briefly summarizes the results and provides some concluding remarks.

2 Related work

The research in this paper primarily uses two pieces of background knowledge: Framework of Core Security Artefacts approach and Delay Tolerant Network.

2.1 Framework of Core Security Artefacts approach

Recently the security requirement analysis has drawn people's attention with the rapid

development both in the software requirements engineering field and in the software security field [Van Lamsweerde, 2004; Foster, 2005; Crook et al., 2005; Jürjens, 2005; Lancy and Nuseibeh, 2004; Lin et al., 2003; Mead, 2005; Moffett et al., 2004; Stavridou, 1998; Viega, 2005; Weber, 2005]. Jürjens uses an extensible verification framework for verifying UML models for security requirement [Jürjens, 2005]. Van Lamsweerde presents an approach to the modeling, specification, and analysis of application-specific security requirements by constructing an intentional anti-model [Van Lamsweerde, 2004]. Foster and Stavridou apply the safety engineering techniques of HAZOP to security requirements development [Foster, 2005; Stavridou, 1998]. Ren et al. suggest a connector-centric architectural approach to derive and argue the completeness of the security requirements [Ren et al., 2005]. Mead and others have proposed a methodology called Security Quality Requirements Engineering (SQUARE) to derive security requirements [Mead and Stehney, 2005]. Weber investigated the completeness analysis of the software security requirements by using a Flaw Taxonomy [Weber et al., 2005]

Framework of Core Security Requirements Artefacts approach (FCSRA) is a security requirements analysis technique proposed by Moffett, Haley, and Nusiebeth [Moffett, 2004]. It develops a systematic method to derive software security requirements. This approach analyzes the functional requirements applying Jackson's Problem Frame, and then decomposes the original Problem Frame into sub-problems. Jackson suggests that although the decomposition can be based on different use-cases, it is not enough to decompose the Problem Frame into sub-problem frames [Jackson, 2001]. The results of applying a Problem Frame approach are the detailed descriptions of the interacting phenomenon between the machine (software) and the domain (software environment) to illustrate the behavior of the software. It gives the users the opportunity to focus on smaller problems and develop detailed requirements specifications. Note that the authors point out that this

FCSRA approach does not have to use the Problem Frame; but that the Problem Frame is the best technique that they are aware of.

To argue the correctness and the completeness of the derived security requirements specification, FCSRA uses the correctness arguments and vulnerability analysis to argue that the security requirement will fulfill the security goals [Moffett et al., 2004]. The correctness argument, first suggested by Jackson in his Problem Frame [Jackson, 2001], is a description that convinces others the proposed machine ensures the requirement is satisfied in the problem domain. Nevertheless, the authors point out that one of the difficulties in assessing the correctness argument is the lack of a methodical approach to discover security vulnerabilities [Moffett et al., 2004]. Harley argues the validity of the correctness arguments by using a structured argument [Harley, 2005]. Other work includes discovering vulnerabilities by focusing on trust assumptions and considering the consequences of failure of the assumptions [Harley, 2004; Lancy, 2004].

2.2 Delay Tolerant Network and Bundle Protocol

Extreme network communication environments are characterized by long signal propagation delays, on the order of seconds, minutes, or hours rather than milliseconds; frequent and lengthy interruptions in connectivity; low levels of traffic coupled with high rates of transmission error; limited bandwidth and highly asymmetrical data transferring rates with a much higher data return rate then command rate [Cert et al., 2005; Scott 2005; Symington, 2005; Burlegigh, 2005; Ramada, 2005; Fall, 2005; Warthman, 2003].

Existing Internet protocols do not work well for the above environments since these environments violate the underlying assumptions on which the Internet architecture is built. These underlying assumptions are not necessarily true for DTNs. According to [Warthman, 2003], the underlying assumptions are:

- During the communication session an end-to-end path exists between source and destination

- The maximum round-trip time over one particular path is not highly variable from packet to packet
- The end-to-end packet and data loss is relatively small
- All routers and end stations support the TCP/IP protocols
- Applications need not worry about communication performance
- Endpoint-based security mechanisms are sufficient for meeting most security concerns

The Bundle Protocol and the Linklider Transmission Protocol are proposed to handle transferring messages in network environments with violations of the above assumptions [Scott et al., 2005; Symington et al., 2005; Ramadas et al., 2005].

The bundle layer proposed in the Bundle Protocol is the end-to-end message-oriented overlay, a layer above the transport layers of the networks and below applications. This overlay employs persistent storage at DTN nodes, includes a hop-by-hop transfer of reliable delivery responsibility, and has optional end-to-end acknowledgement [Scott et al., 2005]. The Licklider Transmission Layer serves as a transport layer with the similar functionality of a TCP/IP layer. However, it provides retransmission-based reliability over links characterized by extremely long message round-trip times and/or frequent interruptions in connectivity [Ramsdas et al., 2005]. Note that in different outer space environments, DTN can choose to use different transport protocols, for example TCP/IP or Licklider. A convergence layer is proposed to serve as the layer between the bundle layer and the transport layer for the purpose of converting different signals from different transport layers to unique signals for the bundle layer or vice versa [Cerf et al., 2005].

A security mechanism needs to be designed to protect the already-limited DTN infrastructure from unauthorized access and usage. The solution cannot be the frequent distribution of a large number of certificates and encryption keys across DTN as is done in today's Internet because of the high-delay and rare resource and scheduled connections. The Bundle Protocol authentication

solution is proposed in [Durst, 2002; Symington, 2005] including the following two points:

- Only first-hop routers need to cache per-user credential and information and only for adjacent users. The credential information includes the user's ID, public/private key, class of service level, and expiration time.
- Downstream routers can rely on the authentication of upstream outers to verify the credential of bundle messages.

This solution provides not only the advantage of storage savings but also an improvement to system management.

This section has provided two pieces of the background knowledge used in this work: Framework of Core Security Requirements Artefacts approach and Bundle Protocol. We show the details of our approach in Section 3 and show how we applied the Framework of Core Security Requirements Artefacts approach on the Bundle Protocol and the results in Section 4.

3 Approach

Figure 12 gives an overview of the analysis technique developed in this paper. It is modified from the Framework of Core Security Requirements Artefacts approach (FCSRA) [Moffett et al., 2004]. The boxes in Figure 12 are the process steps and the artifacts and the lines are the flow of data. The dotted lines and boxes represent the extended bi-directional analysis [Feng and Lutz, 2005; Lutz and Woodhouse, 1997]. The process steps are listed in Figure 13. Note that Figure 12 also illustrates the relationship among the steps and the artifacts by showing how the artifacts interact with each other. For example, the combination of the functional requirements resulting from Step 1.2, the security requirements and the Problem Frame resulting from Step 3, and the correctness arguments and trust assumptions from Step 4.1 is the input to Step 4.2 bi-directional analysis. The output of Step 4.2 is the discovery of the incompleteness of the security requirements and of

remediation, which is input to the Problem Frame and the correctness arguments. In this section we introduce the approach used to analyze the security requirements of the Bundle Protocol.

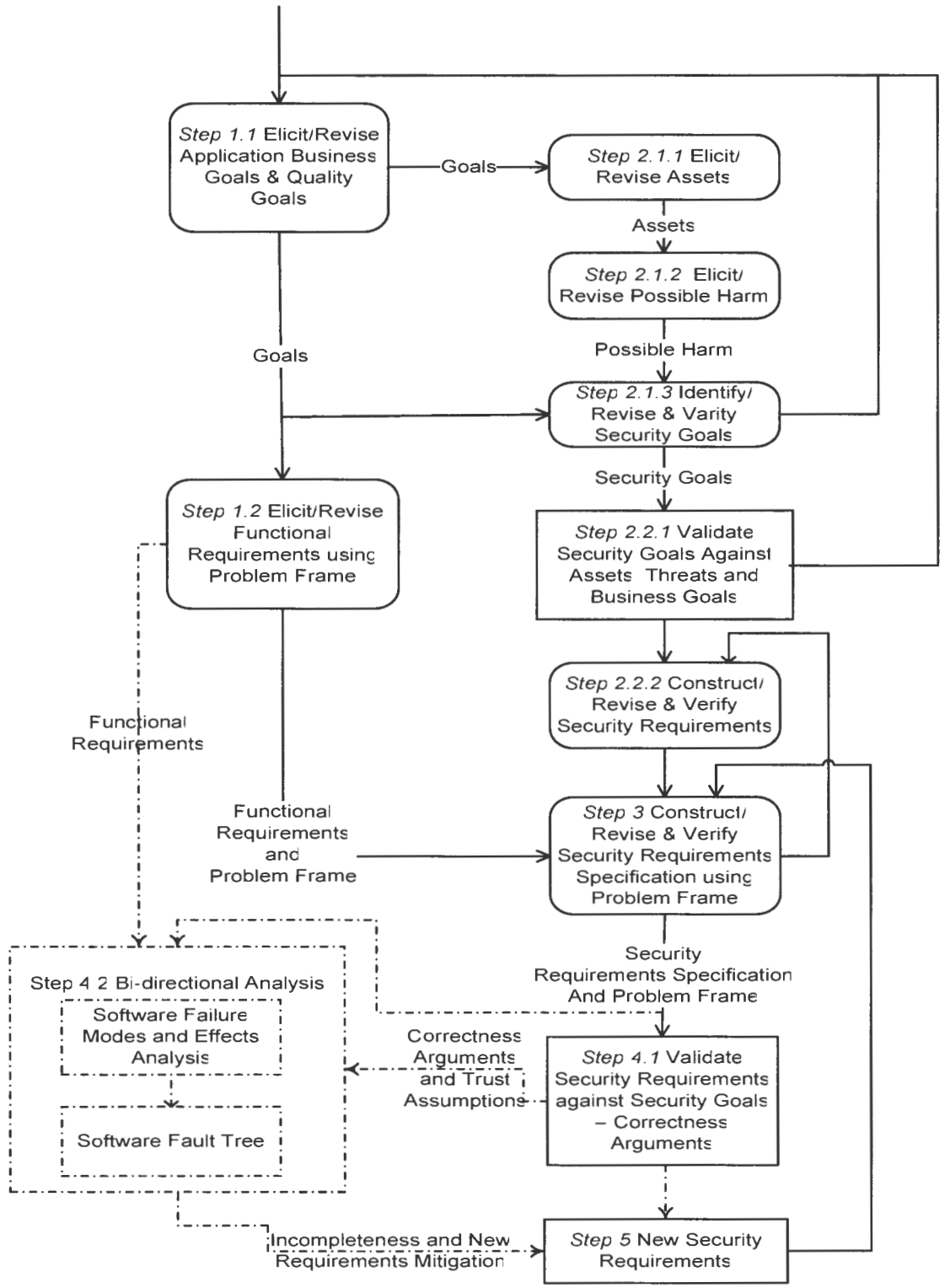

Figure 12. Process Overview [Modified from Moffett et al., 2004]

Step 1: System Business Goals and Functional Requirements
Step1.1 System Business Goals
Step1.2 Functional Requirement:
Using Problem Frame to decompose the requirements and generate the requirements specification.
Step 2: From System Security Goals to System Security Requirements
Step 2.1 Security Goals System Security Risk Analysis (Assets & Harm)
Step 2.1.1.Asset Identification
Step 2.1.2 Harm Identification
Step 2.1.3 Security Goals
Step 2.2 From Security Goals to Security Requirements
Step 2.2.1 Validate Security Goal
Step 2.2.2 Derive Security Requirements
Step 3: from System Security Requirements to Software Security Specifications
Step 3.1 Adding constraints (security requirements) to the machine specification or the properties of other domains
Step 3.2 Altering the interactions between domains
Step 3.3 Introducing additional domains to the problem.
Step 4: Prove the correctness of the Software Security Specifications
Step 4.1 Correctness arguments
Step 4.2 Bi-directional analysis
Step 5: Improved Software Security Specification and improved Problem Frame from the results

Figure 13. Process Steps

4 Results

We apply the improved Framework of Core Security Requirements Artefacts approach shown in Figure 12 and Figure 13 to the Bundle Protocol. The results are listed step by step.

Step.1. Identify System Business Goals and Functional Requirements

This step identifies the business goals and functional requirements for the Bundle Protocol.

Step.1.1. System business goals

From the Delay Tolerant Network Document [Cerf et al., 2005] and the Bundle Layer Document [Scott et al., 2005], we have concluded the following system business goal for the Bundle Layer:

Goal1: performs the application layer and forms a store-and-forward overlay network layer to provide interoperable communications with and among performance-challenged environments where

continuous end-to-end connectivity cannot be assumed [Scott et al., 2005].

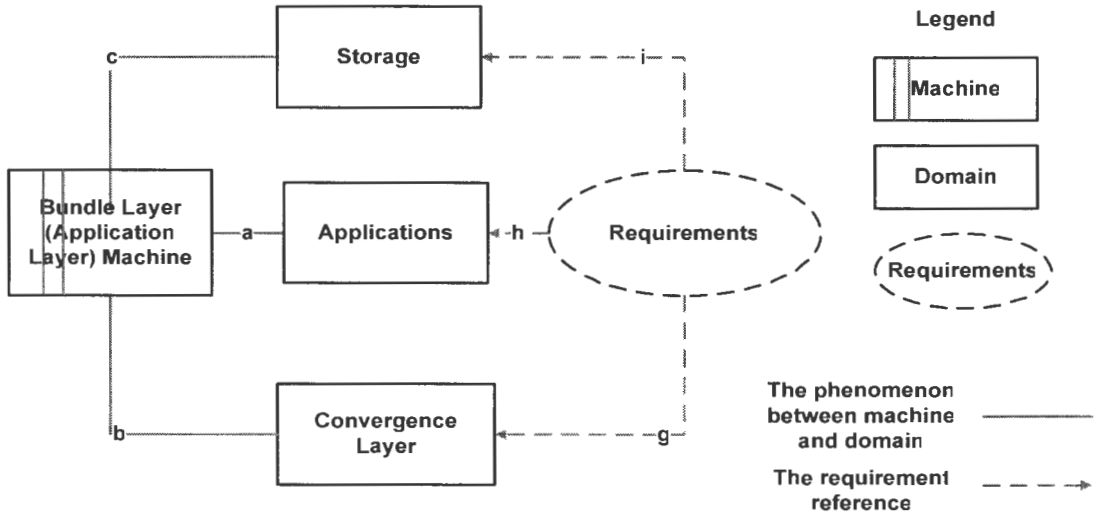

Figure 14 Bundle Protocol Problem Frame

Step.1.2. Functional requirement – using Problem Frame to decompose the requirements

In this step we derive the initial Problem Frame and decompose it into sub-problem frames.

Step.1.2.1. Initial Problem Frame for Bundle Layer Machine

Through the analysis of the documents [Scott et al., 2005; Cert at al, 2005], we have discovered three domains that are interacting with the bundle layer and derived a Problem Frame for the Bundle Protocol shown in Figure 14. The rectangle box with two vertical lines at the left of the figure represents the Bundle Layer machine. The rectangle boxes without vertical lines are the problem domains, representing the environment with which the bundle layer machine interacts. The domains that the bundle layer are interacting with are the Applications (the software applications in a router), the Convergence Layer domain (the layer between the bundle layer and the transport layer), and Storage (the storage that stores the bundles and can be accessed by the bundle layer). The solid lines a, b, and c connecting the domains and the machine represent the interacting phenomenon (signals and events) that occur between the machine and other domains. For example the line "a" between

Bundle Layer Machine and Application domain is the interface between them and the phenomenon on this line. The following "a1" and "a2" belong to the phenomenon "a" [Scott et al., 2005; Symington et al., 2005]:

a1: the applications send sending-bundles requests to the bundle layer to request the sending of bundles; the bundle layer accepts/rejects the requests.

a2: the bundle layer sends data.indicating signals to the applications to indicate the arrival of the data whose destinations are the current routers; the applications accept or reject the data.indicating signals.

The oval shape represents the requirements. The dashed lines "g", "h", and "i" between the requirements and the domains represent the requirements references. Each dotted line corresponds to some requirement on one or more domains. The functional requirements describing the behaviors of the Bundle Layer Machine can then be mapped to those reference lines. Note that in the Problem Frame, the behavior of the machine is represented only by the interface phenomenon with its environment [Jackson, 2001]. For example, one piece of the requirements, that the storage shall store the bundles transferred from the bundle layer, can be mapped to the reference line h.

Step.1.2.2. Decomposition of Problem Frame

The complexity of the interfaces and references makes it hard to see the required relationships between the machine and the problem domains. A good decomposition of the original problem not only can help capture the requirements and describe them but also understand them and analyze them. A way to decompose the problem is to take advantage of use cases and events, although sometimes only using use-cases is not enough to decompose it [Jackson, 2001].

For a typical router in a Delay Tolerant Network (DTN) there are two main use cases involved with the bundle layer's bundle process [Scott et al 2005; Symington et al., 2005]. The first one is that an application in the router initializes a sending-bundle request to the bundle layer and the

bundle accepts the request and forwards the bundle to one of its neighbor in its routing table. Note that in this work we do not focus on the bundle routing issue, although we have fully realized the importance of routing. We name this use case as SENDING. A more detailed description for this use case is:

- The applications of a node in DTN first issues a sending request to the bundle layer
- Then the bundle layer checks the request to see if it can be granted
- If the request is granted, the corresponding bundle will be processed and will either be stored and sent out later when the connection is available or sent out immediately
- If the request is not granted, the bundle layer will send an indicator to the application

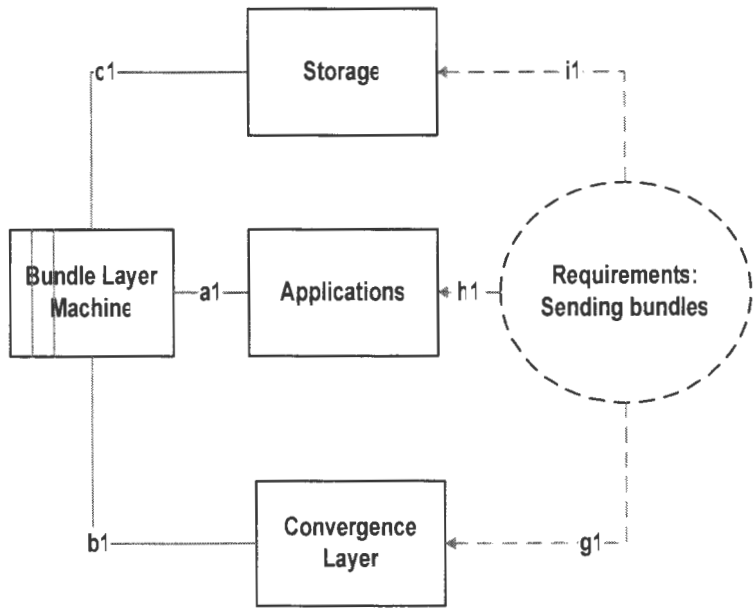

Figure 15. Sub-Problem Use Case SENDING Problem Frame

The second use case is that the bundle layer accepts and forwards an incoming bundle from the convergence layer. We name the second use case as FORWARDING. A more detailed description for this use case is:

- A bundle arrives at the bundle layer from the convergence layer; the bundle will be authenticated first

- If the bundle's owner is recognized and other credential information is matched the bundle layer will accept it, store it in the storage, and forward out later
- If the bundle's owner is not recognized or other credential information is not matched, the bundle layer will not accept it

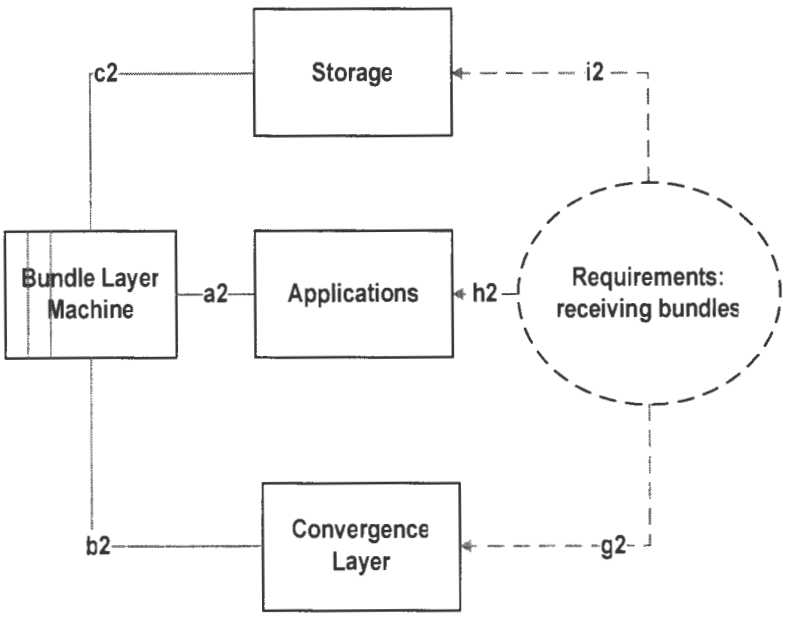

Figure 16. Sub-Problem: Use Case FORWARDING Problem Frame

We decompose the original Problem Frame in Figure 14 into two sub-Problem Frames according to the two use cases and show the resulting sub-Problem Frames in Figure 15 and Figure 16. Although the resulting machine and the domains in the two Problem Frames appear to be the same as the original one, they are different in the machine interfaces with other domains. For example the original line "a" is decomposed into two lines "a1" and "a2". The line a1 in Figure 15 represents the phenomenon that the Application domain sends out a bundle-sending request to the bundle layer machine and the bundle layer either accepts or rejects it. The line "a2" in Figure 16 is the phenomenon that the bundle layer transfers the data indicating acknowledgement to the applications and the applications either accepts or reject it.

Each of the sub-problems can be decomposed into even smaller sub-problems according to the

scenarios and events. For example, the use case FORWARDING can be decomposed into a Bundle Credential Checking Before Accepting sub-problem, an Accepting and Storing Bundles sub-problem, a Routing sub-problem, a Bundle Forwarding sub-problem, and a Local Bundle Delivering sub-problem.

Step.1.2.3. Requirements specification for sub-problems

After the decomposition of the problem, the requirements specification can be derived. The requirements specification is described in a formal specification language as suggested in Framework of Core Security Requirement Artefacts approach [Moffett et al., 2004; Jackson, 2001] and Moffett et al's early work [Moffett et al., 1996]. The typical formats of this specification language are

format 1: MachineName! {Signals and Events} Shall Cause DomainName! {Signals and Event}

format 2: DomainName! {Signals and Events} Shall Cause MachineName! {Signals and Event}.

Format 1 suggests that the signals and the events initialized by the machine shall cause the signals and the events of the domain. Format 2 suggests that the signals and the events initialized by the domain shall cause the signals and the events of the machine.

Figure 17 shows a piece of the functional requirements specification for the use case SENDING and Figure 18 shows a piece of the requirement specification for the use case FORWARDING.

The requirement specification "b2" for the use case FORWARDING describes the interface phenomenon of the Bundle Layer Machine and the Convergence Layer Domain: when a bundle is transferred from the Convergence Layer Domain to the Bundle Machine, it shall result in the following actions of the bundle layer:

- The bundle layer shall do a Bundle Credential Validation for the incoming bundles.
    - Bundle authentication checking. If the authentication fails, the bundle shall be discarded immediately; or else it shall be processed in the following step.

- If the class level of service that the bundle has requested is higher than the class level of service that the node has recorded, it shall be discarded immediately; or else continue the following step.
- If the bundle has expired, the bundle shall be discarded immediately; or else it shall be processed in the following step.

- If the incoming bundle requests a bundle reception status report or the current node is the destination of the bundle, the bundle layer machine shall generate the bundle reception status and send out to the bundle's owner.
- If the bundle's class of service requests custodial transfer or the bundle's destination is the current node, the bundle layer shall transfer the bundle's custody to itself and become the bundle's owner.
- The bundle layer machine shall store the bundle into storage.
- If the bundle's destination is not the current node, the bundle layer machine shall check the routing table and find out the next available chance to forward the bundle.

```
a1:   Application!{<Signal> Send.request (source communications endpoint ID, destination
                    communications endpoint ID, report communications endpoint ID, class of
                    service, delivery options, send token binding, application data unit)
          }
      shall cause
      BLM!{ (<Event> Check Credential(the Bundle) ==Correct)=>
                    (<Event> Initiate bundle transmission procedures,
                     <Signal> SendToken.indication (send token binding, token)
                     <Signal> SendError.Indication
                     )
             )
            or
             (<Event> Check Credential(the Bundle) ==Incorrect)=>
                      (<Event>  Discard the data,
                       <Signal> SendError.Indication
                       )
             )
           }
```

Figure 17. Functional Requirements Specification of Use Case SENDING

```
b2:   Convergence Layer! {<Signal> bundle}
      Shall Cause
      BLM! {(<Event> Authentication (the Bundle) and
               <Event>Class of Service Level Match(the Bundle) and
               <Event> Expiration Checking (the Bundle)
              )
               Shall Cause
             (
               (the incoming bundle.request (bundle reception status report) or the incoming
                  bundle.destination == the current node) =>
                       <Signal> a bundle reception status report
                  Bundle.request (class of service requests custodial transfer) =>
                       <Event> Bundle custody transfers.
                  <Event>Routing Check -> if good to go <Event> forwarding
              )
             ! (<Event> Authentication (the Bundle) and
               <Event>Class of Service Level Match (the Bundle) and
               <Event> Expiration Checking (the Bundle)
              )
               Shall Cause
             (<Event> Discard (the Bundle))
      }
```

Figure 18. Functional Requirements Specification of Use Case FORWARDING

```
a1:   Applications!{<Signal> Send.request (source communications endpoint ID, destination
          communications endpoint ID, report communications endpoint ID, class of
          service, delivery options, send token binding, application data unit)
      }
      shall cause
      BLM!{<Event> Authentication
           <Event>Time Stamp Checking
           <Event>Class of service Checking
      }
```

Figure 19. Functional Requirements Specification for the Credential Validation Event of Use Case SENDING

We continue to decompose the two use cases according to the events in each of the use cases. We focus on two events "a1.1" and "b1.1". The event "a1.1" is the initial of step of the use case SENDING: the credential validation of the outdoing bundles. Similarly the event "b1.1" is also the initial step in the use case FORWARDING. The reason we focus on these is that, because of the limited resources in Delay Tolerant Network to prevent the message flooding attack, the messages

should be discarded as soon as possible if they are from non-authorized entities (i.e. the initial processes should be the most important and main firewall of the security mechanism to prevent the message flooding attack). We show "a1.1" in Figure 19.

Note that although FCSRA has defined that Step 1 only analyzes the functional requirements, in our application we have included the security requirement – the credential validation. The reason is that the Bundle Protocol specification document, on which this work is based, has included the security requirements in it [Scott et al., 2005]. The functional requirements and the security requirements have been integrated together as a whole and cannot be separated. Furthermore, because the authentication is a standard security technique that is used widely for the network, we view it as a functional requirement for the Bundle Layer Machine.

Step.2. From System Security Goals to System Security Requirements

Step 1 has decomposed the bundle layer machine into sub-problems and sub-sub-problems and generated the functional requirements specifications for each interface phenomenon of the Machine and the Domains. Step 2 focuses on the security requirements and modifies the original Problem Frames and functional requirements specification given the security requirements as the constraints of the Problem Frames.

Step.2.1. Security Risk Analysis (Assets & Harm)

In this step, we define the assets and harms.

Step.2.1.1. Asset Identification

We identified the parts of the assets that are uplinked commands to spacecraft and downlinked science date and images – bundles.

Step.2.1.2. Harm Identification

The followings list a part of the hazards identified in this work.

H1: Unavailability

H2: Unauthorized altering;

H3: Unauthorized disclosure.

Because of the limited resources of the Delay Tolerant Network, the unavailability of the service is even more likely to happen than in the Internet. Also because of the importance of the bundles, the unavailability may cause more damage than it does in the Internet. Thus, in this work we are only interested in H1 – unavailability. Furthermore, routers' denial-of-service is one of the harms in the unavailability category, which can be caused by message flooding.

Step.2.1.3. Security Goals

We identified one of the security goals – SG1:

SG1: Preventing messages flooding attack.

Step.2.2. From Security Goals to Security Requirements

First each security goal is examined for possible relevance, and then is analyzed. Finally, the security requirements are derived from the security goals. We only consider the SG1 security goal in our research and from it we derive the following requirements:

SG1/SR1: The bundles shall provide credential information to the bundle layer when it arrives at the bundle layer.

SG1/SR2: The credential information of a user shall be this user's unique identity and nobody else shall be able to get information.

SG1/SR3: The bundles from authorized user or nodes shall be accepted; the bundles from non-legitimated users or nodes shall not be accepted.

SG1/SR4: The bundles from authorized users or nodes shall be forwarded out as soon as possible according to the user or the node's class of service.

SG1/SR5: The bundles from legitimate applications in the current node shall be sent out as soon as possible according to its class of service; the bundles from non-legitimate applications shall

not be sent out.

SG1/SR6: When too many bundles arrive at the bundle layer, the bundle layer shall be able to handle them and shall be able to accept and forward the bundles from legitimate users.

Step.3. From System Security Requirements to Software Security Specifications

In this step, the security requirements are introduced into the Problem Frame. At the same time, the Problem Frame itself may or may not be changed because of the constraints put on the system to fulfill the security goals. Furthermore, from the analyses of the Problem Frame, we obtain detailed security requirements specifications.

Step.3.1. Adding constraints to the machine specification or the properties of other domains.

Four security constraints to the bundle layer machine and other domains are derived from the SG1/SR1 – SG1/SR6.

Constraint 1. Before any client (an entity who uses the DTN) uses the Delay Tolerant Network service or any satellite can join the network to provide the service, it shall transfer its credential information (ID, public/private key, expiration time, maximum class of services) to the adjacent satellites that will communicate with it.

Constraint 2. When any bundle is passed from the convergence layer to the bundle layer or from the applications in the router to the bundle layer, the bundle layer shall first check the bundle credential at first. If the credential information is not correct, the bundle shall be discarded immediately.

Constraint 3. The credential information stored in the storage shall be unable to be accessed by any other applications in the routers, except for the bundle layer.

Constraint 4. The only bundles stored in storage shall be the bundles with correct credential.

Step.3.2. Altering the interactions between domains and introducing additional domains to the problem.

We modified the initial Problem Frame as a result of applying the four constraints listed above. The new Problem Frame is shown in Figure 20. We have added two additional domains in the Problem Frame. One of them is another storage called Credentials Storage, which stores the credentials for the clients and the neighbor satellites. The Credential Checker Domain is a domain decomposed from the Bundle Layer Machine aiming at checking credentials. The Credential Storage can only be accessed by the Credential Checker Domain.

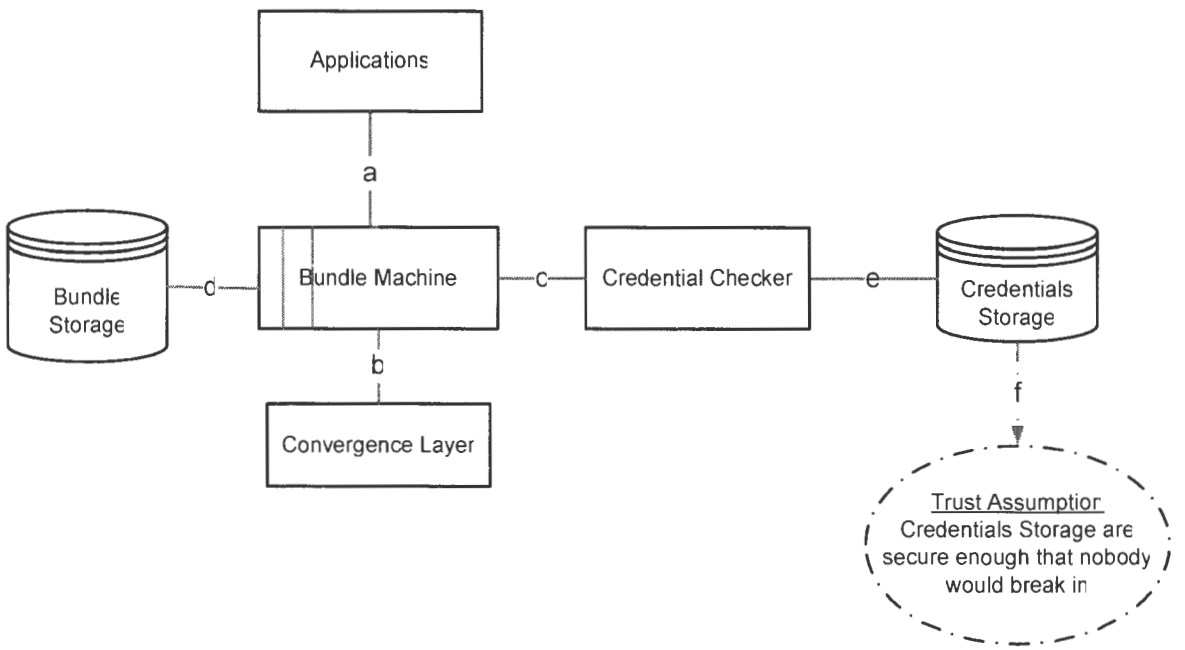

Figure 20. Problem Frame for Bundle Protocol with Security Requirements

In addition to the new domains, a trust assumption is put on the Credential Storage domain that the data stored in the credential storage is secured and will not be able to be accessed by a third party. Trust Assumptions are the claims about the behavior or the membership of domains included in the

system, where the claims are made in order to satisfy a security requirement. [Harley et al. 2004]. Figure 21 is a piece of the security requirements specification for the new Credential Storage.

We claim that the new Problem Frame satisfies the four constraints listed in the step 3.1.

```
b:    Convergence Layer!{ <Signal> Bundle}
      Shall Cause
      Bundle Machine ! {
            <Event> Defragment  (Message) and obtain the bundle header
            <Signal> the bundle header
            <event> Credential Checking
      }
e:    Credential-Checking!{<Signal> an ID and the class of the service}
      Shall Cause
      Credential Storage {
            if (existing(ID) in the Credential Database ) =>
            {<Signal> User Public Key, <Signal>User Credentials,
            <Signal>timestamp }
            else
            {<Signal> NO }
e:    Credential Storage! {No}
      Shall Cause
      Credential-Checking Machine!{
                  <Event> Discard the Bundle immediately,
                  <Signal> NO}
e:    Credential Storage!{
            <Signal> User Public Key,
            <Signal>User Credentials,
            <Signal>timestamp }
      Shall Cause
      Credential Checking Machine! {
            <Event> CheckValid (User Public Key, User Credentials, Signature,
            Destination, Class of Service)
            Shall Cause
            <Event> Pass the bundle to the Bundle Processing Domain
            else shall cause
            <Event> Discard the bundle
       }
```

Figure 21. Security Requirements Specification for Credential Validation for the Incoming Bundles

Step.4. Prove the correctness of the Software Security Specifications

The goal of Step 4 is to assess the correctness and the completeness of those security specifications (i.e., they would fulfill the security goal). Since we only focus on the goal SG1 of

preventing the message flooding attacks, this section only discuss the correctness arguments related to SG1.

Step.4.1. Correctness arguments

The correctness argument claims that the Problem Frame's domains, interactions and specification will satisfy the system requirements [Jackson, 2001]. It includes a positive argument and a negative argument. The positive argument attempts to argue why the Problem Frame satisfies the requirements; the negative argument tests the Problem Frame by searching for contradictions to the argument [Moffett, 2004]. In the case of security requirements the contradictions are called vulnerabilities. Lamsweerde [Lamsweerde, 2004] describes how a piece of vulnerability can be discovered by negating the requirement and then attempting to show that the negation of the requirement can be satisfied.

The positive correctness argument to argue that the Problem Frame will be able to prevent message flooding attack is the following:

- Only bundles with correct credential are only those bundles that can be stored in the storage

The negative argument is:

- The bundles with incorrect credentials are the only bundles that are discarded.

The above two arguments are true if and only if the following trust assumptions are true:

Trust Assumption *TA1*: The users will not abuse their credentials in the Delay Tolerant Network (i.e., they will not launch a message flooding attack).

Trust Assumption *TA2*: The credential of a user or a node will not be given or be obtained by a malicious third party.

Step.4.2. Bi-directional Analysis

To assess the correctness arguments the Framework of Core Security Requirement Artefacts approach (FCSRA) [Moffett et al., 2004] suggests that the vulnerability analysis will be a good

option to find weaknesses in a design. At the same time, their other work suggests that the trust assumptions provide a focus point for discovering vulnerabilities by considering the consequences of failure of the assumptions [Harley, 2004; Lancy, 2004].

In the work we use the extended bi-directional analysis to launch a vulnerability analysis. The bi-directional analysis combines a forward analysis (from failure modes to effects) with a backward analysis (from hazards to contributing causes). It starts with Software Failure Modes and Effects Analysis (SFMEA) and finds out what are the failure effects if any of the events and data is in failure modes. Then the failure modes are taken as the root nodes of fault trees [Lutz and Woodhouse, 1997]. In this work we extended the bi-directional analysis by taking the result of the Problem Frame and the security requirements specification as its input and analyzing each of the events and the signals one the machine and domain interfaces. This provides a systematic structure to start the correctness analysis and the assurance of maximum coverage of the problem.

Step.4.2.1. Software Failure Modes and Effects Analysis

Software Failure Modes and Effects Analysis (SFMEA) analyzes the failure modes of the data and events and their corresponding effects. Usually the data and events input to the SFMEA are derived from software architectural design: the data are the data interacting between the components and the events are the events that occur in the components. In this work we extend bi-directional analysis by taking the signals and events from the interfaces of the machine as inputs. An interface is an area where domains and machine connect and communicate. The data used in the SFMEA are the signals transmitted through the interface; the events used in event table are the events that occur on the interface.

Table 6: A piece of Event Table in the Software Failure Mode and Effect Analysis

Event	Event Fault Type	Description	Local Effect	End Effect
After the incoming bundle has been authenticated and its destination is not the current node, the bundle layer stores the bundle into the storage and wait for the next best chance to forward the bundle out.	Halt /Abnormal Termination	E1. The process of storing the bundle is stuck, hung, or deadlocked.	The bundle is not stored to the storage domain.	Denial-of-Service
	Omission	E2. The store of the bundle is omitted.	The bundle is not stored;	Denial-of-Service
	Incorrect Logic /Event	E3. The bundle is stored into an incorrect storage Location. E4. The actual storage position is not matching the storage position that the bundle is recorded.	The bundle is not stored in the storage location that the bundle layer recorded as.	Denial-of-Service
	Timing /Order	E5. The store of the bundle is delayed.	The bundle is not stored when the bundle layer is trying to access it.	Denial-of-Service
		E6. The store of the bundle is done previous the authentication of the bundle.	Non-authenticated bundle could be stored.	Denial-of-Service
		E7. The storing occurs repeatedly rather than regularly.	The storage does not have enough resources to process the following bundles.	Denial-of-Service
		E8. The requesting of the bundle storing is so often that when the next requesting of the storing is happening, the storage is not ready to receive it and process it.		

Table 7: A piece of Data Table in the Software Failure Mode and Effect Analysis

Data Item	Data Fault Type	Description	Local Effect	End Effect
Bundles passed from the bundle layer to the storage	Absent Data	D1. The bundle is not transferred from the bundle layer to the bundle storage.	The bundle is not stored into the storage	Denial-of-Service
	Incorrect Data	D2. When the bundle is passed from the bundle layer to the bundle storage, the bundle is altered incorrectly.	Incorrect bundles are stored.	Denial-of-Service
	Timing of Data Wrong	D3. When the bundles are transferred from the bundle layer machine to the storage, the bundle is delayed for a long period of time.	When the bundle layer tries to search the bundle in the storage, it could not find it or only obtain wrong data.	Denial-of-Service
		D4. The bundle layer machine transfers unexpected amount of bundles to the storage.	The storage can be occupied by spurious bundles.	Denial-of-Service
		D5. The bundle is stored in the storage so long that it is expired.		Denial-of-Service
	Duplicate Data	D6. Redundant copies of bundles are stored in the storage.	The storage becomes unavailable because it is full.	Denial-of-Service

We show a piece of the event table in Table 6. Here the event is "after the incoming bundle has been authenticated and its destination is not the current node, the bundle layer stores the bundle into the storage and waits for the next best chance to forward the bundle". It occurs between the bundle layer machine and the storage domain. For example, one failure mode "E1" of the event is "the process of storing bundles is stuck, hung, or deadlocked", which leads to the local effect: "the bundle is not stored", and then eventually results in the denial of service to this bundle.

Table 7 shows a piece of the data table in the software failure modes and effect analysis. The data is bundles that have been recognized by a legal resource and shall be stored in the storage. For example when the data is in an "absent" failure mode, the local effect is "the bundle is not stored into the storage" and the end effect is denial-of-service to this bundle.

Step.4.2.2. Software Fault Tree Analysis

Any of the failure modes in the SFMEA analysis can be the root node of a software fault tree. Through the fault tree analysis the reason of these failure modes can be discovered and remediation can be achieved.

In Figure 22, we show a piece of the Software Fault Tree analysis result. The root of this fault tree is the failure mode D3: when the bundles are transferred from the bundle layer machine to the storage, the bundle's storing is delayed for a long period of time. The result shows four faults in the level 7 – L7.1, L7.2, L7.3, and L7.4 related to the message flooding that could cause the denial of service. L7.1 and L7.3 are that the clients and the routers are sending too many messages to the Delay Tolerant Network, which violates the trust assumption TA1. L7.2 and L7.4 are that the compromised clients and the routers are sending too many messages out to the Delay Tolerant Network, which violates the trust assumption TA2.

Step.4.2.3. Remediation

There are several ways to prevent the hazard in the root of the fault tree from happening, shown

as in Figure 22. They include:

1. Stop the occurrence of the faults on the lowest levels in the fault tree. There are two approaches:
    a. The clients and the routers use better security techniques to prevent them from being hacked and compromised. However this approach is not related to the Bundle Protocol security requirements. Furthermore, it is really hard to know how secure the nodes shall be that hackers would not break in.
    b. Control the rate at which the clients and the routers can inject bundles into the Delay Tolerant Network traffic. There shall be a maximum rate such that the clients and hops can send out bundles and forbid the routers or clients from sending bundles when the rate exceeds the maximum rate.
2. Stopping one child fault from leading to the parent fault in the fault tree:
    a. Stop the fault L4.2 from leading to the fault L3.2 (i.e., when the bundle layer receives an overwhelming number of bundles with a rate that it cannot handle, the bundle layer shall be able to discard the incoming bundles accordingly). Leveson points out that the completeness criteria for requirements analysis includes ensuring the capacity or load of the software: "a minimum and maximum load assumption must be specified for every interrupted-signaled event whose arrival rate is not dominated (limited) by another type of event" [Leveson, 1995].
3. Detecting the faults and doing the necessary corrections before it is too late:
    a. Prevent the fault L1.2 (i.e., there shall be a response when the storage is full).. Leveson suggests that "the response to excessive inputs (violations of load assumptions) must be specified" [Leveson, 1995].

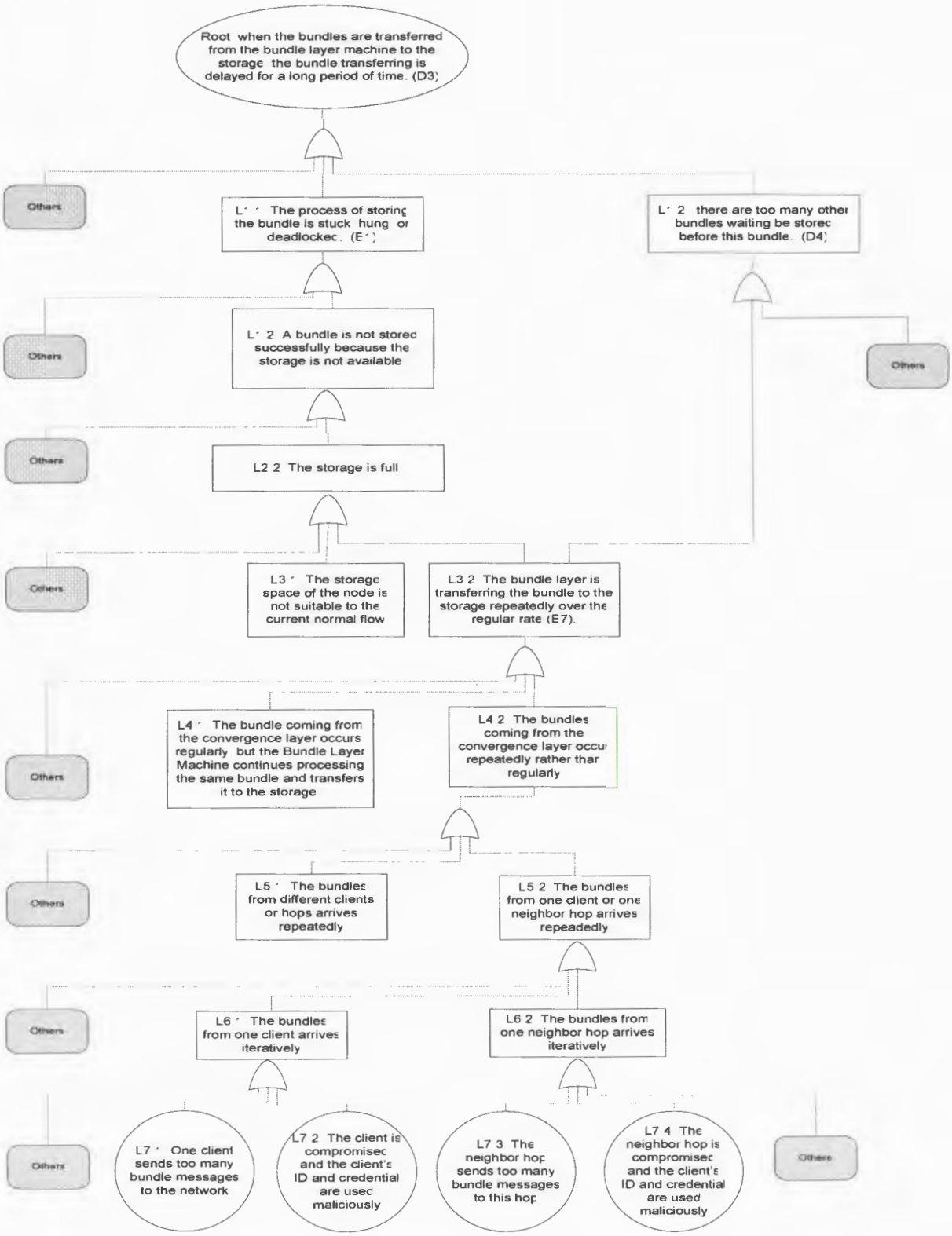

Figure 22. One Piece of Software Fault Tree for Bundle Protocol

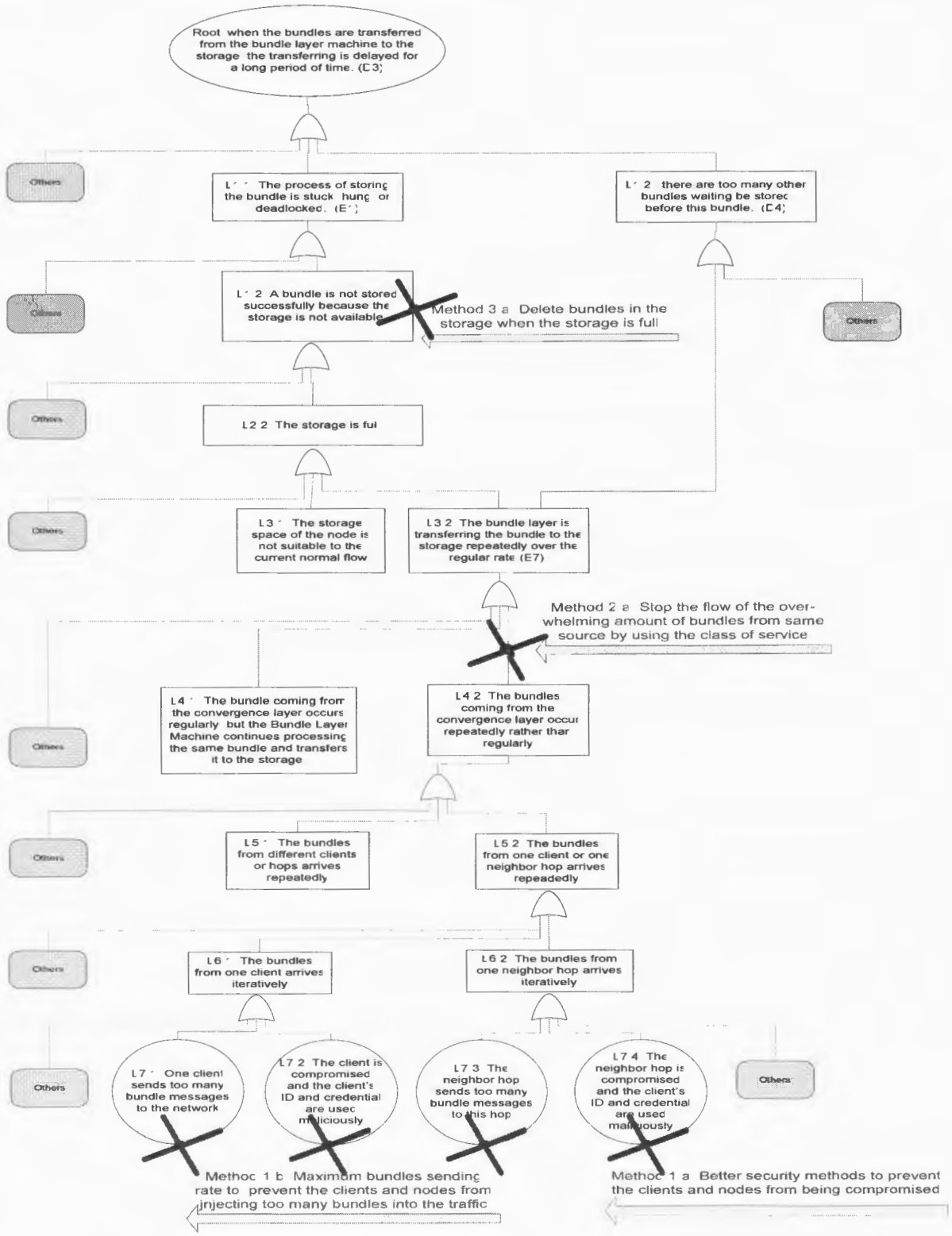

Figure 23. A Piece of Software Fault Tree for Bundle Protocol with Remediation

We propose the following remediation for the Bundle Protocol requirements according to the discussion above:

Remediation 1. The class of service information in the credential shall include the maximum rate at which the clients and the routers inject bundles into the Daley Tolerant Network traffic.

Remediation 2. The Bundle Layer Machine shall be able to detect the receiving rates from a client or a neighbor router. If the client or the neighbor hop transfers too many messages within a time period exceeding the maximum rate, the machine shall be able to block the communicating channel between itself and the client or the neighbor routers.

Remediation 3. The bundle layer shall be able to detect the rate at which it injects bundles into the Deley Tolerant Network traffic. When the rate exceeds the maximum rate that is defined in its class of service level, it shall be able to adjust the injections.

Remediation 4. The bundle layer shall be able to detect the availability of the storage space and delete some bundles when the storage is running out of space.

Remediation 5. The clients who use the Delay Tolerant Network and the satellites in the Delay Tolerant Network shall be certified and accredited regarding to their ability to keep their security levels.

Note that the Flow Control and Congestion Control in the Delay-Tolerant Network are similar to the four piece of requirements proposed above [Cerf et al., 2005]. They have been mentioned briefly in the document and the authors suggest they have not "reached complete consensus" for the two controls and have "unresolved concerns" about them. We also found that the Bundle Protocol security document has not mentioned the Flow Control and Congestion Control at all [Scott et al., 2005].

Step.5. Improved Software Security Specification and Improved Problem Frame

After further analysis of the four pieces of the additional requirements and the Problem Frame,

the revised Problem Frame is shown in Figure 24. We have added two more domains to the Problem Frame: Bundle Storage Availability Checker and Bundle Transferring Rate Checker. In addition, the four pieces of remediation proposed previously have been mapped to the requirement references in Figure 24.

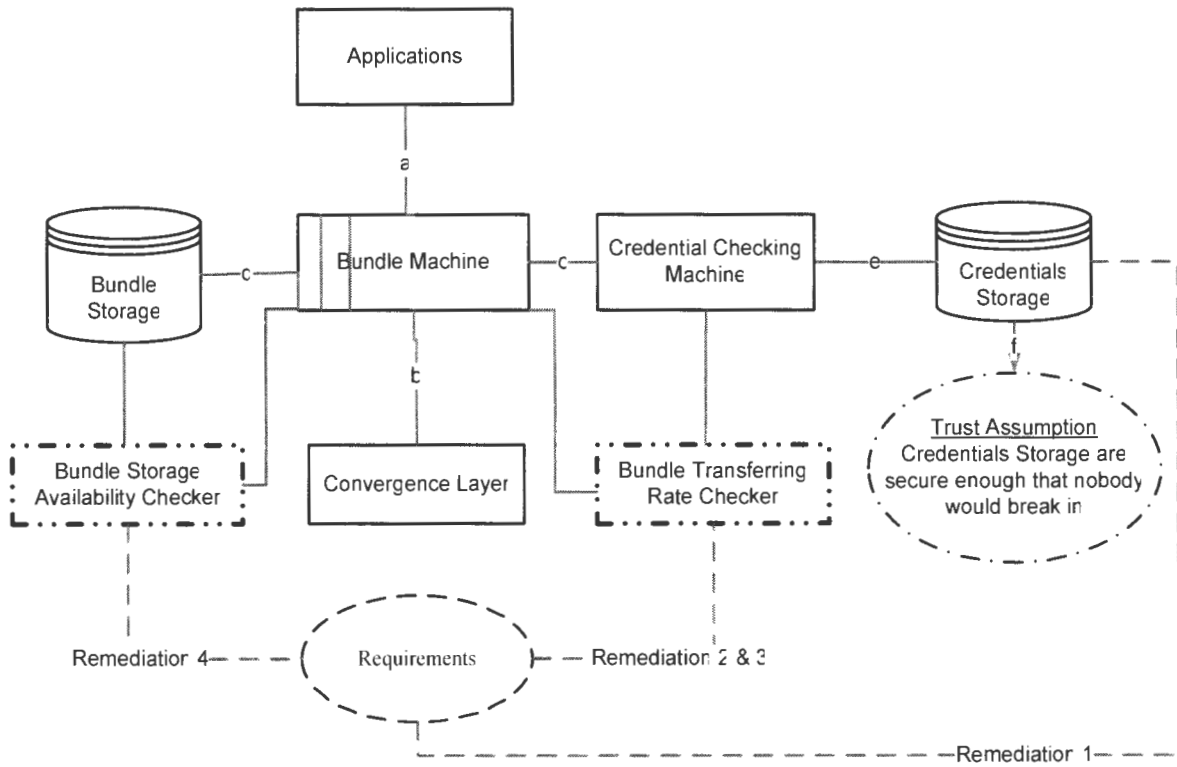

Figure 24. Improved Problem Frame for Bundle Protocol After

The new Problem Frame and the remediation have strengthened the trust assumptions *TA1* and *TA2* list in Step 4.1 and at the same time weakened the dependence of the correctness argument on the trust assumption. Thus, we believe that the validity of the correctness arguments has been strengthened using this approach.

5 Discussion

In the previous section we have presented the results of the improved Framework of the Core Security Requirement Artefact Approach. In this section we give a discussion of some issues that we encountered in this research, including the benefits of using Framework of the Core Security Requirement Artefact Approach, the incompleteness and the ambiguousness of the Delay Tolerant Network documents, the complexity of the class of service level, and the bundle authentication options.

5.1 The benefits of using Framework of the Core Security Requirement Artefact Approach

Framework of the Core Security Requirement Artefact Approach (FCSRA) is the main approach used in this work. Below, we discuss the reasons for using FCSRA.

First, this approach uses Problem Frame method, which only analyzes the problem/requirements instead of the solution/architecture. Our purpose is to perform analysis on the security requirements and we do not want to include the information of the architectural design. Most of the previous research on security requirement analysis is based on the architectural design. This was described in Section 2. However, since the architectural design is based on the requirements specification and it is the result of complete and correct requirements, we want to avoid using it to assess the completeness and the correctness of the requirements. The Problem Frame approach gives us another option in that we can only consider the problem (i.e., requirements) instead of the solution (i.e., architectural design), although we discovered in our research that the Problem Frame and the architectural design are connected in some degree and some work has been done to derive the Problem Frame from the architectural design [Rapanotti, 2004].

Secondly, using Problem Frame to analysis the security requirements is easy to trace and reuse in the future. The Problem Frame approach decomposes the requirements into detailed requirements

specifications without the result of the architectural design. The ability of the Problem Frame's decomposition provides the traceability and the reusability requirements specifications generated from the Problem Frame. Due to the quick development of hardware and software, the Delay Tolerant Network (DTN) is going to be a rapid evolving system, which will benefit from the traceability of the requirements including security requirements'. The decomposition of the problem (requirements) gives the big benefit of an easy requirement tracing and reusability. In addition, this approach will contribute to the reuse of the DTN software. In outer space, the environments are unpredictable, which demands different protocols. For example, the network layer can use TCP/IP protocol or LickLider protocols: in earth environment, TCP/IP will be used; in highly delayed environment, LickLider protocol is going to be used [Scott et al., 2005]. This results in a protocol family. To achieve reuse easily, correctly, and widely, the decomposed problem and the corresponding security requirements specifications provide a better and fundamental way.

However, this approach has a weakness. After the security requirements specification is constructed, the approach tries to argue its correctness and completeness by using "correctness arguments". The correctness argument is unable to convince the users that the resulting security requirements specification can fulfill the security goal. It fails to provide convincing data and fails to give a complete coverage of the problem. Bi-directional analysis has provided a way to address the weakness above and gives the extended Framework of the Core Security Requirement Artefact Approach more power.

5.2 Incompleteness and ambiguousness of the Delay Tolerant Network documents

During our research we have discovered some difficulties in understanding the documents about Delay Tolerant Network and its protocols specifications. They have several instances of incompleteness and ambiguity. For example, the Flow Control and Congestion Control are mentioned in the Delay Tolerant Network Architecture document [Cert et al., 2005], but not in the

Bundle Protocol Security Specification document [Symington et al., 2005].

One example of ambiguity is that in the Bundle Layer Protocol document [Scott et al., 2005], credential registration is not mentioned although the registration is necessary to achieve the security goal. The ambiguity is whether the bundle layer handles the credential registration of the clients and other neighbor routers or not.

If it is not the bundle layer that is responsible for the credential registration, then some application in the routers shall handle it. Then suppose a bundle of registration targeting the current router arrives at the bundle layer. The kind of header this message will carry to let the bundle layer recognize it as a registration bundle and continue to pass it to the applications is unclear. This header will be some kind of administration header, and the registration application must be able to access the Credential Storage, which increases the vulnerability of being compromised and increases the level of damage that a compromised router can cause the Delay Tolerant Network.

Otherwise, if the Bundle Protocol is responsible for the registration, the bundle with the registration information needs some special header that would invoke the bundle layer's registration process. This special header shall carry some administration keys too, but at the same time there is no need for a registration application to access the Credential Storage.

With the consideration of the high security requirement of the credential storage we believe the bundle layer should be responsible for the registration. However, there are still a lot of issues regarding this, and we have included this in our future work.

5.3 The complexity of Class of Service Level

In Step 4.2.3, four pieces of additional requirements are proposed for remediation. The first remediation is that the class of service information in the credential shall include the maximum rate by which the clients and the hops can inject bundles into the Delay Tolerant Network traffic. We further propose that the bundle layer can limit the total number of messages of a client or a hop

transferred within a period of time, instead of rate. For example, the maximum might be five messages per day for some client. Satellites with better memory and located in important positions (for example, gateways) shall have higher level of messages numbers that could be transferred per day. In addition, different outer space environments could have different range requirements. At the same time the node will have to judge the priorities of the messages and send out the messages that are most important and schedule to delay those messages whose priorities are low. However, different hops might adopt different orders to send out the messages

5.4 Bundle Layer Authentication Options

The Bundle Protocol authentication solution is:

- Only first-hop routers need to cache per-user credential and information and only for adjacent users
- Downstream routers can rely on the authentication of upstream routers to verify the authenticity of bundle messages

However, this approach is partially susceptible to compromised routers. Security infrastructures are specially needed to detect and prevent the malicious actions from the compromised routers and the compromised clients. For example, if an otherwise-legitimate router is compromised, it would be able to utilize network resources to send traffic purportedly originating from any user whose identity is known to the route [Durst, 2002; Symington, 2005].

However, if a message signature is carried end-to-end (an option for DTN security) instead of being replaced by the hops', a legitimate user could repudiate the origin of any traffic generated in this manner. Thus, a reasonable trade-off is to admit the possibility that a compromised router could launch a denial-of-service attack in order to gain the scalability benefits of not checking end-user credentials at every hop.

6 Summary

This work provides a structured method and step-by-step guidelines for deriving a security analysis from Problem Frames in order to improve the protocol security. This Chapter has presented a systematic method for performing security analysis on a network protocol. The method extended the Framework of Core Security Artefacts Analysis with bi-directional safety analysis. The bi-directional analysis is modified by using the Problem Frame result as input instead of architectural design. The results of the Framework of Core Security Artefacts are a Problem Frame, security requirements specifications, and the correctness arguments that demonstrate the correctness and completeness of the security requirements specifications. The bi-directional analysis checks the validity of the correctness argument, questions the trust assumptions, and looks for the vulnerabilities of the protocol requirements. Bi-directional analysis provides a way to make the FCSA complete. Findings from application of the bi-directional safety-analysis method included four new security-related requirements.

In this work, we take an initial step towards the completeness and the correctness analysis of the security requirements of the Bundle Layer Protocols. Currently we are investigating the reusability of the analysis result. Furthermore, some interesting events (for example the credential registration) are also planned as future work. We hope our research will motivate more study towards application of bi-directional analysis both on the software safety analysis and software security analysis.

REFERENCES

Allenby, K., Kelly, Trim., 2001. Deriving Safety Requirements Using Scenarios: Requirements Engineering. In: Proceedings of the Fifth IEEE International Symposium, Toronto, Ont., Canada, pp. 228-235.

Ardis, A. M., Weiss, D. M., 1997. Defining Families: The Commonality Analysis, Software Engineering. In: Proceedings of the 1997 (19th) International Conference, pp. 649–650.

Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Paech, B., Wust, J., Zettel, J., 2002. Component-based Product Line Engineering with UML, Addison-Wesley, 2002.

Bass, L., Clements, P., Kazman, R., 2003. Software Architecture in Practice, Addison-Wesley.

Bayer, J., Flege, O., Gacek, C., 2000. Creating Product Line Architectures. In: Proceedings of the 3rd International Workshop on Software Architectures for Product Families. Lecture Notes in Computer Science Volume 1951, Springer-Verlag, pp. 210-216.

Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., 1999. PuLSE: A methodology to develop software product lines. In: Proceedings of the 5th Symposium on Software Reusability, Los Angeles, CA, USA, pp. 122-131.

Bosch, J., 2000. Design and Use of Software Architectures. Addison-Wesley.

Burgess, M., 2003. Fault Tree Creation and Analysis Tool: User Manual. Available from http://www.iu.hio.no/FaultCat. (Accessed July 2004).

Burleigh, S. Ramadas, M. Farrell, S. 2005. Licklider Transmission Protocol – Motivation. Available from http://www.dtnrg.org/docs/specs/draft-irtf-dtnrg-ltp-motivation-01. (Accessed Oct. 2005).

Cerf, V. Burleigh, S. Hooke, A. Durst, R. Scott, K. Fall, K. Weiss, H. V. 2005. Delay Tolerant Network Architecture. Available from http://www.dtnrg.org/docs/specs/draft-irtf-dtnrg-arch-03.txt. (Accessed Oct. 2005).

Clauß, M. 2001. Modeling variability with UML. In GCSE 2001Young Researchers Workshop, Erfurt, Germany.

Cook, D. J., Huber, M., Gopalratnam, K., Youngblood, M., Learning to Control a Smart Home Environment. Available from http://ranger.uta.edu/~holder/courses/cse6362/pubs/Cook03.pdf. (Accessed July 2004).

Crook, D., Lin. L., Nuseibeh. B. 2002. Security Requirements Engineering: When Anti-requirements Hit the Fan , Proceedings of IEEE International Requirements Engineering Conference (RE'02), Essen, Germany, 9-13 September 2002.

Dehlinger, J., Lutz, R., 2004. Software Fault Tree Analysis for Product Lines. In: Proceedings of the 8th IEEE International Symposium on High Assurance Systems Engineering, Tampa, FL, USA, pp. 12-21.

Demmer, E. Brewer, K. Fall, S. Jain, M. Ho, R. Patra, R. 2004. Implementing Delay Tolerant Network. Available from http://www.dtnrg.org/papers/demmer-irb-tr-04-020.pdf. (Accessed Oct. 2005).

Demmer, M. 2005. DTN Reference Implementation. Available from http://www.dtnrg.org/docs /presentations/IETF60/dtn-impl-ietf-8-6-04-demmer.pdf. (Accessed Oct. 2005).

Doerr, J., 2002. Requirements Engineering for Product Lines: Guidelines for Inspecting Domain Model Relationships. Diploma Thesis, University of Kaiserslautern.

DTD tutorial. Availabel from http://www.w3schools.com/dtd/. (Accessed July 2004).

Durst, B. 2002. Infrastructure Security Model for Delay Tolerant Networks. Available from http:// www.dtnrg.org/docs/papers/dtn-sec-wp-v5.pdf. (Accessed Oct. 2005).

Egyed, A., Mehta, N. R., Medvidovic, N., 2000. Software Connectors and Refinement in Family Architectures. In: Proceedings of the Third International Workshop on Development and Evolution of Software Architectures for Product Families (ARES III). Lecture Notes in

Computer Science Volume 1951, Springer-Verlag, pages 96-105.

Eriksson, Henrik; Jönsson, Patrik Implementation and analysis of the bundling protocol for delay-tolerant network architectures http://epubl.ltu.se/1402-1617/2005/139/LTU-EX-05139-SE.pdf.

Fall, K. 2003. A Delay-Tolerant Network Architecture for Challenged Internets. Available from http://www.intel-research.net/Publications/Berkeley/030320031146_120.pdf. (Accessed Oct. 2005).

Fall, K. 2004. Messaging in Difficult Environments presented to the IETF/IAB Workshop on Messaging, Oct 2004. Available from http://www.dtnrg.org/papers/kfall-irb-tr-04-019.pdf. (Accessed Oct. 2005).

Farrell, S. Ramadas, M. Burleigh, S. 2005. Licklider Transmission Protocol - Extensions. Available from http://www.dtnrg.org/docs/specs/draft-irtf-dtnrg-ltp-extensions-01.txt. (Accessed Oct. 2005).

Fellbaum, K., Hampicke, M., Human-Computer Interaction in a Smart Home Environment. Available from http://www.sentha.tu-berlin.de/paper/fm_ha_miami.pdf. (Accessed July 2004).

Feng, Q., Lutz, R.R. 2005. Bi-Directional Safety Analysis of Product Lines. Journal of Systems and Software, Volume 78, 2005, pp.111-127.

Foster, N. L. 2005. The Application of Software and Safety Engineering Techniques to Security Protocol Development Available from http://citeseer.ist.psu.edu/article/foster03application.html. (Accessed Oct. 2005).

Goseva-Popstojanova, K., Hassan, A., Guedem, A., Abdelmoez, D. W., Ammar, H., 2003. Architectural-Level Risk Analysis Using UML. IEEE Transactions on Software Engineering, pp. 946-960.

Haley, C. B., Moffett, J. D., Laney, R., Nuseibeh, B., 2005. Arguing Security: Validating Security Requirements Using Structured Argumentation , Proceedings of the 3rd Symposium on

Requirements Engineering for Information Security (SREIS'05) held in conjunction with the 13th International Requirements Engineering Conference (RE'05), Paris, France, 29 August 2005.

Haley, C. B., Laney, R. C., Moffett, J. D., Nuseibeh, B., 2004. "The Effect of Trust Assumptions on the Elaboration of Security Requirements," in Proceedings of the 12th International Requirements Engineering Conference (RE'04). Kyoto Japan: IEEE Computer Society Press, 6-10 Sep 2004, pp. 102-111.

Haley, C., Laney, R., Nuseibeh, B., 2004. Using Problem Frames and Projections to Analyze Requirements for Distributed Systems, Proceedings of 10th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'04), Riga, Latvia, Essener Informatik Beiträge, 7-8 June 2004.

John, I., Muthig, D., 2002. Tailoring Use Cases for Product Line Modeling. In: Proceedings of International Workshop on Requirements Engineering for Product Lines, Essen, Germany, pp. 26-32.

Jürjens, J. 2005. Security: Sound methods and effective tools for model-based security engineering with UML. In Proceedings of the 27th International Conference on Software Engineering. May 2005.

Kang, K. C., Kim, S., Lee, J, Kim, K., Shin, E., Huh, M., 1998. FORM: A feature-oriented reuse method with domain-specific reference architectures. Annals of Software Engineering, Volume 5, 1998, pp. 143-168.

Lamsweerde, V. A. 2004. Elaborating Security Requirements by Construction of Intentional Anti-Models. In Proceedings of the 26th International Conference on Software Engineering May 2004

Landwehr, C. E., Heitmeyer, C. L., McLean, J. 1984. A Security Model for Military Message Systems ACM Trans. Comput. Syst. 2(3): 198-222.

Laney, H. R., Nuseibeh, B., Deriving Security Requirements from Crosscutting Threat Descriptions. In Proceedings of 3rd International Conference on Aspect Oriented Software Development (AOSD '04), Lancaster, UK, 22-26 March 2004, ACM Press.

Leveson, N. G. 1986. Software Safety: Why, What, and How. ACM Computing Surveys, Vol. 18, No. 2, pp. 25-69.

Leveson, N. G., 1995. Software System Safety. Addison-Wesley.

Lin, B. Nuseibeh, D. Ince, M. Jackson, and J. Moffett, Introducing Abuse Frames for Analysing Security Requirements , Poster Paper, Proceedings of 11th IEEE International Requirements Engineering Conference (RE'03), 371-372, Monterey, USA, 8-12 September 2003.

Lu, D., Lutz, R., 2002. Fault Contribution Trees for Product Families. In: Proceedings of the 13th International Symposium on Software Reliability Engineering, pp. 231-242.

Lutz, R. R., 2000. Extending the Product line Approach to Support Safe Reuse. The Journal of Systems and Software, Vol. 53, Issue 3, pp. 207-217.

Lutz, R. R., Gannod, G. C., 2003. Analysis of a Software Product Line Architecture: An Experience Report. The Journal of Systems and Software, vol. 66: 3, pp. 253-67.

Lutz, R. R., Woodhouse, R. M., 1997. Requirements analysis using forward and backward search. Annals of Software Engineering, Vol 3, pp. 459 –475.

MavHome. Available from http://mavhome.uta.edu/. (Accessed July 2004).

Mead, N. R. Stehney, T. 2005. Security Quality Requirements Engineering (SQUARE) Methodology. In proceedings of the Workshop on Software Engineering for Secure Systems – Building Trustworthy Applications (SESS'05) St. Louis, Missouri, USA.

Moffett, J.D., Hall, J., Coombes, A., McDermid, J. 1996. A Model for a Causal Logic for Requirements Engineering. Journal of Requirements Engineering, 1996. 1(1): p. 27-46.

Moffett, J. D. Haley, C. B. Nuseibeh, B. 2004. Technical Report No 2004/23 Core Security Requirements Artefacts. Available from http://computing-reports.open.ac.uk/index.php/2004/200423. (Accessed Oct. 2005).

Nentwich, C., 2002. Xlinkit Version 5 Language Reference. Available from http://www.systemwire.com/xlinkit/Downloads/xlinkit-5.3/doc/reference/reference.html. (Accessed July 2004).

Nentwich, C., Capra, L., Emmerich, W., Finkelstein, A. F., 2002. xlinkit: A Consistency Checking and Smart link Generation Service. ACM Transactions on Internet Technology, Vol. 2, pp. 151-185.

Perry, D. E., 1998. Generic architecture descriptions for product lines. In: Proceeding Of ARES II: Software Architectures for Product Families (LNCS 1429), Springer-Verlag, pp. 51-56.

Ramadas, M. Burleigh, S. Farrell, S. 2005. Licklider Transmission Protocol – Specification. Available from http://www.dtnrg.org/docs/specs/draft-irtf-dtnrg-ltp-03.txt. (Accessed Oct. 2005).

Rapanotti, L., Hall, J., Jackson, M., Nuseibeh, B. 2004. Architecture Driven Problem Decomposition, In Proceedings of 12th IEEE International Requirements Engineering Conference (RE'04), Kyoto, Japan, 6-10 September 2004.

Ren, J., Taylor, R., Dourish P., Redmiles D. 2005. Towards An Architectural Treatment of Software Security: A Connector-Centric Approach. In proceedings of the Workshop on Software Engineering for Secure Systems – Building Trustworthy Applications (SESS'05) St. Louis, Missouri, USA.

Sabanosh, T. Sullivan, K., 2001. Interoperability for Probabilistic Risk Assessment Tools Using Domain-Specific Markup Languages. Available from http://dependability.cs.virginia.edu /bibliography/Interoperable.pdf. (Accessed July 2004).

Scott, K. Burleigh, S. 2005 Bundle Protocol Specification Available from http://www.dtnrg.org/docs/specs/draft-irtf-dtnrg-bundle-spec-03.txt (Accessed Oct. 2005).

Smart Home–project. Institute Electronics at Tamperre University of Technology, Finland. Available from http://www.ele.tut.fi/research/personalelectronics/projects/smart_home.htm. (Accessed July 2004).

Srivatanakul, T., Clark, J. A., Polack, F. 2004. Effective Security Requirements Analysis: HAZOP and Use Cases. Lecture Notes in Computer Science. In Proceedings of Information Security: 7th International Conference, ISC 2004, Palo Alto, CA, USA, Sep. 27 – 29, 2004. pp416 – 427.

Stavridou, V. Dutertre, B. 1998. From Security to Safety and Back. Computer Security, Dependability, and Assurance: From Needs to Solutions. P. 182.

Symington, S. 2005. DTN Security. Available from http://www3.ietf.org/proceedings/05mar /slides/DTNRG-4/dtnrg-4.ppt#302,2,DTN Security. (Accessed Oct. 2005).

Symington, S. Farrell, S. Weiss, H. 2005. Bundle Security Protocol Specification. Available from http://www.dtnrg.org/docs/specs/draft-irtf-dtnrg-bundle-security-00.txt (Accessed Oct. 2005)

Sommerville, I. 2004. Software Engineering. Addison-Wesley.

Tine V., Piessens F., Win B. D., Joosen W. 2005. Requirements Traceability to Support Evolution of Access Control. In proceedings of the Workshop on Software Engineering for Secure Systems – Building Trustworthy Applications (SESS'05) St. Louis, Missouri, USA.

Viega, J. Building Security Requirements with CLASP. 2005. In proceedings of the Workshop on Software Engineering for Secure Systems – Building Trustworthy Applications (SESS'05) St. Louis, Missouri, USA.

Warthman, F. 2003. Delay-Tolerant Networks (DTNs) A Tutorial. Available from http://www .ipnsig.org/reports/DTN_Tutorial11.pdf. (Accessed Oct. 2005).

Weber S., Karger P., Paradkar A. 2005. A Software Flaw Taxonomy: Aiming Tools at Security. In proceedings of the Workshop on Software Engineering for Secure Systems – Building Trustworthy Applications (SESS'05) St. Louis, Missouri, USA.

Weiss, D. M., Lai, C. T. R., 1999. Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley.

Youngblood, G. M., 2003. Mavhome Research Scope. Available from http://www.aimachines.com/youngblood/papers/MVH-2003-2_MavHome_Scope_2003.pdf. (Accessed July 2004).